Pentium® II Processor Specification Update

Release Date: May 13, 1998

Order Number: 243337-014

The Pentium® II processor may contain design defects or errors known as errata which may cause the product to deviate from published specifications. Current characterized errata are documented in this Specification Update.

CONTENTS

REVISION HISTORY

Date of Revision	Version	Description
May 1997	-001	This document is the first Specification Update for the Pentium® II processor.
June 1997	-002	Added Erratum 25. Update Erratum 13 status in the Summary Table of Changes. Added Documentation Change Table and Documentation Change 1. Added 300-MHz Pentium II processor information.
July 1997	-003	Added Erratum 26. Added Specification Change Table and Specification Changes 1 and 2.
August 1997	-004	Added Erratum 27. Added Document Change 2 and Spec Changes 3, 4, 5, 6, and 7.
September 1997	-005	Updated Erratum 27. Added Errata 28 and 29. Added Document Change 3 and Spec Clarification 1. Added C1 stepping information. Updated Spec Change 6.
October 1997	-006	Updated Errata 6 and 18, and S-spec table.
November 1997	-007	Updated Erratum 22. Added Specification Clarification 2, 3, and 4.
December 1997	-008	Updated and added notes to S-spec table. Updated package information table. Updated Erratum 24. Added Errata 30, 31, and 32.
January 1998	-009	Added notes to Pentium II processor markings. Updated Erratum 28. Added Erratum 33. Added Documentation Change 4 and 5. Added Specification Change 5.
January 26, 1998 (Special Edition)	-010	Updated S-spec table. Added dA0 stepping information. Added Errata 34, 35, 36, 37, and 38.
February 11, 1998	-011	Added new processor markings. Corrected Errata 13 and 34 for steppings affected. Corrected typos in summary table for Errata 34, 35, and 36. Added Erratum 39. Added Documentation Change 6.

REVISION HISTORY (Contd.)

Date of Revision	Version	Description
March 11, 1998	-012	Added new boxed processor markings. Updated Documentation Changes section, Specification Clarifications section, and Specification Changes section. Corrected Erratum 8. Added Errata 40, and 41. Added Documentation Changes 6 and 7. Added Specification Clarification 6. Added Specification Changes 1 and 2.
April 8, 1998	-013	Added new Mobile Pentium II processor markings and Pentium II Mobile Modules markings. Updated Documentation Changes section, Specification Clarifications section, and Specification Changes section. Updated S-spec table. Added new steppings to Summary Table of Changes. Corrected Erratum 1. Added Errata 42, 43 and 44. Added Documentation Change 8. Updated Specification Change 1. Added Specification Change 3.
May 13, 1998	-014	Updated S-spec table. Updated Errata 2 and 42. Added Errata 45 through 51. Corrected Documentation Change 7. Updated Specification Change 2.

PREFACE

This document is an update to the specifications contained the *Pentium® II Processor Developer's Manual* (Order Number 243341), the *Pentium® II Processor at 233 MHz, 266 MHz, 300 MHz, and 333 MHz* datasheet (Order Number 243335), the *Mobile Pentium® II Processor* datasheet (Order Number 243669), the *Pentium® II Processor Mobile Module (MMC1)* datasheet (Order Number 243667) and the *Intel Architecture Software Developer's Manual, Volumes 1, 2 and 3* (Order Numbers 243190, 243191, and 243192, respectively). It is intended for hardware system manufacturers and software developers of applications, operating systems, or tools. It contains Specification Changes, S-Specs, Errata, Specification Clarifications, and Documentation Changes.

Nomenclature

Specification Changes are modifications to the current published specifications for the Pentium® II processor. These changes will be incorporated in the next release of the specifications.

S-Specs are exceptions to the published specifications, and apply only to the units assembled under that s-spec.

Specification Clarifications describe a specification in greater detail or further highlight a specification's impact to a complex design situation. These clarifications will be incorporated in the next release of the specifications.

Documentation Changes include typos, errors, or omissions from the current published specifications. These changes will be incorporated in the next release of the specifications.

Errata are design defects or errors. Errata may cause the Pentium II processor's behavior to deviate from published specifications. Hardware and software designed to be used with any given processor stepping must assume that all errata documented for that processor stepping are present on all devices.

Identification Information

The Pentium II processor can be identified by the following values:

Family[1]	233-, 266-, 300-MHz Model 3[2]	233-, 266-, 333-MHz Model 5[2]
0110	0011	0101

NOTES:

1. The Family corresponds to bits [11:8] of the EDX register after RESET, bits [11:8] of the EAX register after the CPUID instruction is executed with a 1 in the EAX register, and the generation field of the Device ID register accessible through Boundary Scan.
2. The Model corresponds to bits [7:4] of the EDX register after RESET, bits [7:4] of the EAX register after the CPUID instruction is executed with a 1 in the EAX register, and the model field of the Device ID register accessible through Boundary Scan.

The Pentium II processor's second level (L2) cache size can be determined by the following register contents:

512-Kbyte Unified L2 Cache[1]	43h

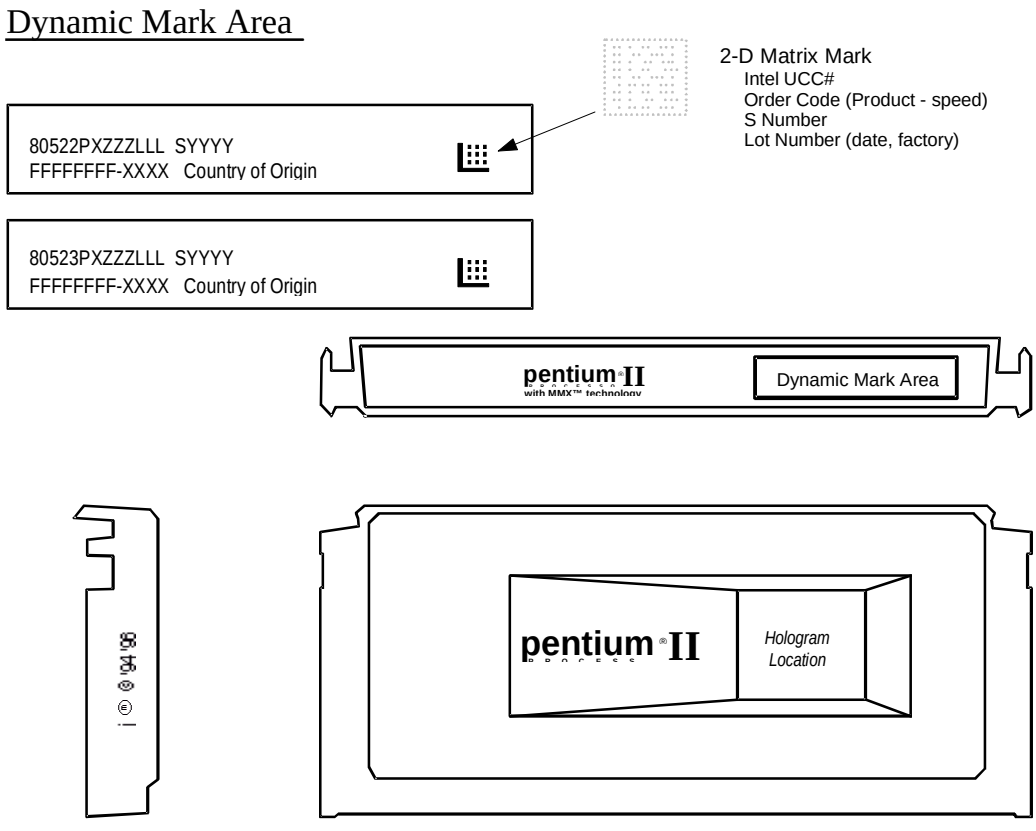
NOTE:

1. For the Pentium® II processor, the unified L2 cache size corresponds to the value in bits [3:0] of the EDX register after the CPUID instruction is executed with a 2 in the EAX register. Other Intel microprocessor models or families may move this information to other bit positions or otherwise reformat the result returned by this instruction; generic code should parse the resulting token stream according to the definition of the CPUID instruction.

Specification Update for Pentium® II Processors

GENERAL INFORMATION

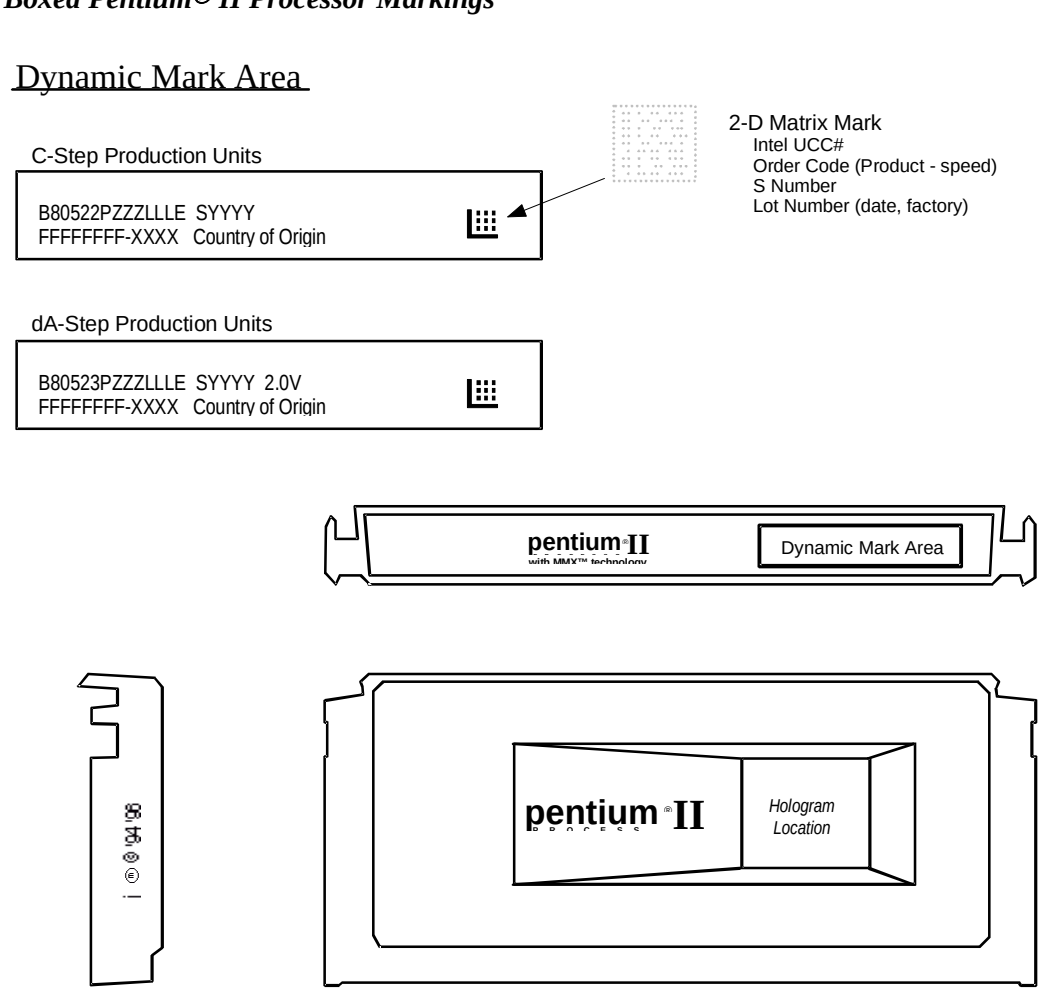
Pentium® II Processor Markings



NOTES:

- ZZZ = Speed (MHz).
- SYYYY = S-spec Number.
- LLL = Level 2 Cache Size (in Kilobytes).
- FFFFFFFF = FPO # (Test Lot Traceability #).
- XXXX = Serialization Code.

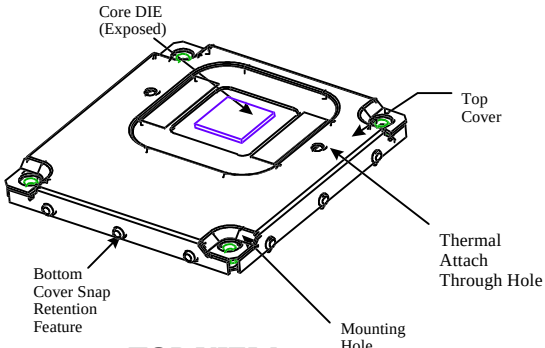
Boxed Pentium® II Processor Markings



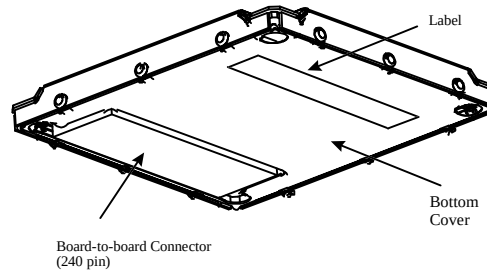
NOTES:

- ZZZ = Speed (MHz).
- LLL = Level 2 Cache Size (in Kilobytes).
- E = ECC Support in Level 2 Cache
- SYYYY = S-spec Number.
- FFFFFFFF = FPO # (Test Lot Traceability #).
- XXXX = Serialization Code.

Mobile Pentium® II Processor Markings



TOP VIEW



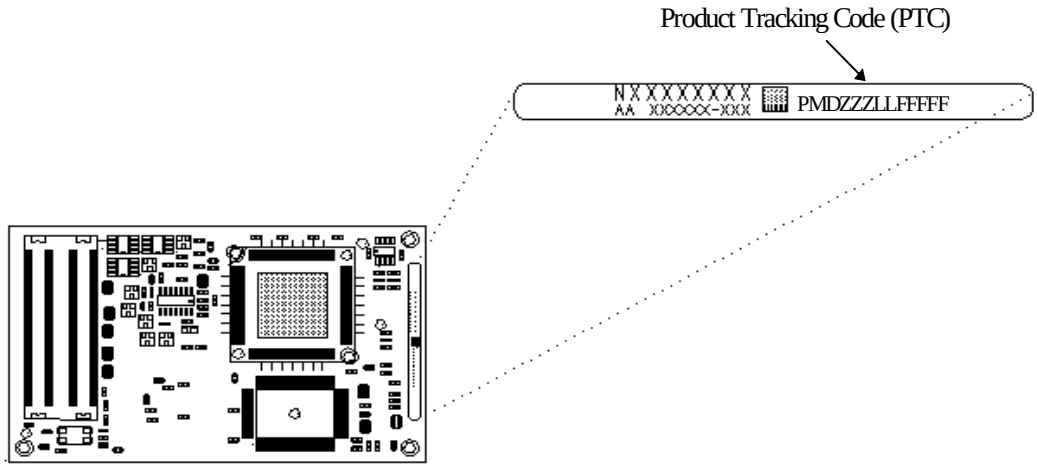
BOTTOM VIEW

Typical Production Mark:

NOTES:

- ZZZ = Speed (MHz).
- SYYYY/QYYY = S-spec Number/QDF Number.
- FFFFFFFF = FPO # (Test Lot Traceability #).

Pentium® II Processor Mobile Modules Markings



NOTES:

- ZZZ = Speed (MHz).
- LL = Cache size code.
- FFFFF = FPO # (Test Lot Traceability #).

Basic Pentium® II Processor Identification Information

CPUID										
Type	Family	Model	Step-ping	Core Step-ping	L2 Size (Kbytes)	TagRAM/ Stepping	S-Spec/ PTC	ECC/ Non-ECC	Speed (MHz) Core/Bus	Notes
0	6	3	3	C0	512	T6/B0	SL264	non-ECC	233/66	1,2
0	6	3	3	C0	512	T6/B0	SL265	non-ECC	266/66	1,2
0	6	3	3	C0	512	T6/B0	SL268	ECC	233/66	1,2
0	6	3	3	C0	512	T6/B0	SL269	ECC	266/66	1,2
0	6	3	3	C0	512	T6/B0	SL28K	non-ECC	233/66	1, 2, 3
0	6	3	3	C0	512	T6/B0	SL28L	non-ECC	266/66	1, 2, 3
0	6	3	3	C0	512	T6/B0	SL28R	ECC	300/66	1,2
0	6	3	3	C0	512	T6/B0	SL2MZ	ECC	300/66	1, 2, 3
0	6	3	4	C1	512	T6/B0	SL2HA	ECC	300/66	1,2
0	6	3	4	C1	512	T6/B0	SL2HC	non-ECC	266/66	1,2
0	6	3	4	C1	512	T6/B0	SL2HD	non-ECC	233/66	1,2
0	6	3	4	C1	512	T6/B0	SL2HE	ECC	266/66	1,2
0	6	3	4	C1	512	T6/B0	SL2HF	ECC	233/66	1,2
0	6	3	4	C1	512	T6/B0	SL2QA	non-ECC	233/66	1, 2, 3
0	6	3	4	C1	512	T6/B0	SL2QB	non-ECC	266/66	1, 2, 3
0	6	3	4	C1	512	T6/B0	SL2QC	ECC	300/66	1, 2, 3

Basic Pentium® II Processor Identification Information (Contd.)

CPUID										
Type	**Family**	**Model**	**Step-ping**	**Core Step-ping**	**L2 Size (Kbytes)**	**TagRAM/ Stepping**	**S-Spec/ PTC**	**ECC/ Non-ECC**	**Speed (MHz) Core/Bus**	**Notes**
0	6	5	0	dA0	512	T6P/A3	SL2KA	ECC	333/66	4, 5, 6
0	6	5	0	dA0	512	T6P/A3	SL2QF	ECC	333/66	3, 4, 5, 6
0	6	5	1	dA1	512	T6P-e/A0	SL2QH	ECC	333/66	3, 4, 6, 7
0	6	5	1	dA1	512	T6P-e/A0	SL2S5	ECC	333/66	4, 6, 7
0	6	5	1	dA1	512	T6P-e/A0	SL2S6	ECC	350/100	4, 6, 7
0	6	5	1	dA1	512	T6P-e/A0	SL2S7	ECC	400/100	4, 6, 7
0	6	5	1	dA1	512	T6P-e/A0	SL2SF	ECC	350/100	3, 4, 6, 7
0	6	5	1	dA1	512	T6P-e/A0	SL2SH	ECC	400/100	3, 4, 6, 7
0	6	5	0	mdA0	512	T6P/A3	SL2KH	ECC	233/66	8
0	6	5	0	mdA0	512	T6P/A3	SL2KJ	ECC	266/66	8
0	6	5	0	mm dA0	512	T6P/A3	PMD233 05001AA	ECC	233/66	9
0	6	5	0	mm dA0	512	T6P/A3	PMD266 05001AA	ECC	266/66	9

NOTES:

1. V_{CC_CORE} is specified for 2.8 V +100/-70 mV for these Pentium® II processors.
2. T_{PLATE} is specified for 5 °C - 75 °C for these Pentium II processors with S.E.C. cartridge packages except for s-specs SL28R , SL2HA, SL2MZ, and SL2QC which have a Tplate specification for 5 ºC - 72 ºC.
3. This is a boxed Pentium II processor with an attached fan heatsink.
4. V_{CCCORE} is specified for 2.0 V +100/-70 mV for these Pentium II processors.
5. T_{PLATE} is specified for 5 °C - 65 °C for these Pentium II processors.
6. The clock requirement for the product under these QDF/S-spec numbers differ from those in the current specification. These differences are as follows:

BCLK Signal Quality Guidelines for Edge Finger Measurement

T#	Parameter	Min	Nom	Max	Unit	Notes
V1':	BCLK V_{IL}			0.5	V	
V2':	BCLK V_{IH}	2.0			V	

Slot 1 Processor System Bus AC Specifications (Clock) at the Processor Edge Fingers

T#	Parameter	Min	Nom	Max	Unit	Notes
T3':	BCLK High Time	3.99 1.70			ns	@>2.0 V [a] @>2.0 V [b]
T4':	BCLK Low Time	4.39 2.37			ns	@<0.5 V [a] @<0.5 V [b]
T5':	BCLK Rise Time	1.02 1.13		2.68 2.94	ns ns	(0.5 V–2.0 V) [a] (0.5 V–2.0 V) [b]
T6':	BCLK Fall Time	1.02 1.13		2.68 2.94	ns ns	(2.0 V–0.5 V) [a] (2.0 V–0.5 V) [b]

NOTES:

a) This specification applies to Slot 1 processors when operating with a Slot 1 processor system bus frequency of 66 MHz.

b) This specification applies to Slot 1 processors when operating with a Slot 1 processor system bus frequency of 100 MHz.

7. For these Pentium II processors:
   - T_{PLATE} is specified for 5 °C - 75 °C with ETP (extended thermal plate).
   - Cacheable address space supports up to 4 GB.
8. This is a Mobile Pentium II processor, with $V_{CC_{CORE}}$ = 1.7 V.
9. This is a Pentium II processor Mobile Module, with $V_{CC_{CORE}}$ = 1.7 V.

Pentium® II Processor Package Information

	Core				
S-Spec/PTC Number	**Stepping**	**Speed (MHz)**	**L2 Size(Kbytes)**	**Processor Substrate Revision**	**Cartridge Revision**
SL264	C0	233	512	D	3.00
SL265	C0	266	512	D	3.00
SL268	C0	233	512	D	3.00
SL269	C0	266	512	D	3.00
SL28K	C0	233	512	D	3.00
SL28L	C0	266	512	D	3.00
SL28R	C0	300	512	D	3.00
SL2MZ	C0	300	512	D	3.00
SL2HA	C1	300	512	D	3.00
SL2HC	C1	266	512	D	3.00
SL2HD	C1	233	512	D	3.00
SL2HE	C1	266	512	D	3.00
SL2HF	C1	233	512	D	3.00
SL2QA	C1	233	512	D	3.00
SL2QB	C1	266	512	D	3.00
SL2QC	C1	300	512	D	3.00
SL2KA	dA0	333	512	B1	3.00
SL2QF	dA0	333	512	B1	3.00
SL2QH	dA1	333	512	B1	3.00
SL2S5	dA1	333	512	B1	3.00
SL2S6	dA1	350	512	B1	3.00
SL2S7	dA1	400	512	B1	3.00
SL2SF	dA1	350	512	B1	3.00
SL2SH	dA1	400	512	B1	3.00
SL2KH	mdA0	233/66	512	A0	1.00
SL2KJ	mdA0	266/66	512	A0	1.00
PMD23305001AA	mmdA0	233/66	512	N/A	N/A
PMD26605001AA	mmdA0	266/66	512	N/A	N/A

Summary Table of Changes

The following table indicates the Specification Changes, Errata, Specification Clarifications, or Documentation Changes which apply to the Pentium II processors. Intel intends to fix some of the errata in a future stepping of the component, and to account for the other outstanding issues through documentation or specification changes as noted. This table uses the following notations:

CODES USED IN SUMMARY TABLE

X:	Specification Change, Erratum, Specification Clarification, or Documentation Change applies to the given processor stepping.
Doc:	Intel intends to update the appropriate documentation in a future revision.
Fix:	This erratum is intended to be fixed in a future stepping of the component.
Fixed:	This erratum has been previously fixed.
NoFix:	There are no plans to fix this erratum.
(No mark) or (blank box):	This item is fixed in or does not apply to the given stepping.
AP:	APIC related erratum.
SUB:	This column refers to errata on the Pentium® II processor substrate.
Shaded:	This erratum is either new or modified from the previous version of the document.

NO.	C0	C1	dA0	dA1	mdA0	mmdA0	SUB	Plans	ERRATA
1	X	X	X	X	X	X		NoFix	FP Data Operand Pointer may be incorrectly calculated after FP access which wraps 64-Kbyte boundary in 16-bit code
2	X	X	X	X	X	X		NoFix	Differences exist in debug exception reporting
3	X	X	X	X				NoFix	FLUSH# servicing delayed while waiting for STARTUP_IPI in 2-way MP systems
4	X	X	X	X	X	X		NoFix	Code fetch matching disabled debug register may cause debug exception
5	X	X	X	X	X	X		NoFix	Double ECC error on read may result in BINIT#
6	X	X	X	X	X	X		NoFix	FP inexact-result exception flag may not be set
7	X	X	X	X	X	X		NoFix	BTM for SMI will contain incorrect FROM EIP
8	X	X	X	X	X	X		NoFix	I/O restart in SMM may fail after simultaneous MCE
9	X	X	X	X	X	X		NoFix	Branch traps do not function if BTMs are also enabled
10	X	X	X	X				NoFix	Checker BIST failure in FRC mode not signaled
11	X	X	X	X				NoFix	BINIT# assertion causes FRCERR assertion in FRC mode
12	X	X	X	X	X	X		NoFix	Machine Check Exception handler may not always execute successfully
13	X	X	X	X	X	X		NoFix	MCE due to L2 parity error gives L1 MCACOD.LL
14	X	X	X	X	X	X		NoFix	LBER may be corrupted after some events
15	X	X	X	X	X	X		NoFix	BTMs may be corrupted during simultaneous L1 cache line replacement
16	X							Fix	System may hang due to internal protocol violation
17	X							Fix	Livelock condition may cause system hang
18	X	X						Fix	Mispredicted branch may cause incorrect tag word on MMX™ technology instructions
19	X	X						Fix	Thermal sensor/THERMTRIP# does not work
20	X	X						Fix	Spurious machine check exception via IFU data parity error

NO.	C0	C1	dA0	dA1	mdA0	mmdA0	SUB	Plans	ERRATA
21	X	X						Fix	Loss of inclusion in IFU can cause Machine Check Exception
22	X	X						Fix	Possible system hang when paging is disabled and reenabled from uncached memory
23	X	X						Fix	L2 performance counters miscount L2_RQSTS
24	X	X						Fix	Erroneous signaling of user mode protection violation
25	X							Fix	Invalid operation not signaled by the FIST instruction on some out of range operands
26	X	X						Fix	FLUSH# assertion disables L2 Machine Check Exception reporting
27	X	X						Fix	EFLAGS may be incorrect after a multiprocessor TLB shootdown
28	X	X	X	X				Fix	Delayed line invalidation issue during 2-way MP data ownership transfer
29	X	X	X	X	X	X		Fix	Potential early deassertion of LOCK# during split-lock cycles
30	X	X	X	X	X	X		NoFix	A20M# may be inverted after returning from SMM and Reset
31	X	X	X	X	X	X		Fix	Reporting of floating-point exception may be delayed
32	X	X						Fix	EFLAGS discrepancy on a page fault after a multiprocessor TLB shootdown
33	X	X	X	X	X	X		NoFix	Near CALL to ESP creates unexpected EIP address
34								Fixed	Deep Sleep exit transition may cause hang
35			X	X	X	X		Fix	Built-in self test always gives nonzero result
36			X	X	X	X		Fix	THERMTRIP# may not be asserted as specified
37			X		X	X		Fix	Cache state corruption in the presence of page A/D-bit setting and snoop traffic
38			X		X	X		Fix	Snoop cycle generates spurious Machine Check Exception
39	X	X	X	X	X	X		Fix	MOVD/MOVQ instruction writes to memory prematurely

NO.	C0	C1	dA0	dA1	mdA0	mmdA0	SUB	Plans	ERRATA
40	X	X	X	X	X	X		NoFix	Memory type undefined for nonmemory operations
41			X	X				NoFix	Infinite snoop stall during L2 initialization of MP systems
42	X	X	X	X	X	X		NoFix	Bus protocol conflict with optimized chipsets
43	X	X	X	X	X	X		NoFix	FP Data Operand Pointer may not be zero after power on or Reset
44	X	X	X	X	X	X		NoFix	MOVD following zeroing instruction can cause incorrect result
45	X	X	X	X	X	X		NoFix	Premature execution of a load operation prior to exception handler invocation
46	X	X	X	X	X	X		NoFix	Read portion of RMW instruction may execute twice
47	X	X	X	X				Fix	Test pin must be high during power up
48	X	X	X	X	X	X		Fix	Intervening writeback may occur during split-lock
49	X	X	X	X	X	X		NoFix	MC2_STATUS MSR has model-specific error code and machine check architecture error code reversed
50	X	X	X	X	X	X		NoFix	Mixed cacheability of lock variables is problematic in MP systems
51					X	X		Fix	Thermal sensor may assert SMBALERT# incorrectly
1AP	X	X	X	X	X	X		NoFix	APIC access to cacheable memory causes SHUTDOWN
2AP	X	X	X	X	X	X		NoFix	2-way MP systems may hang due to catastrophic errors during BSP determination
3AP	X	X	X	X				NoFix	Write to mask LVT (programmed as EXTINT) will not deassert outstanding interrupt

NO.	C0	C1	dA0	dA1	mdA0	mmdA0	SUB	Plans	DOCUMENTATION CHANGES
1	X	X	X	X	X	X		Doc	Invalid arithmetic operations and masked responses to them relative to FIST/FISTP instruction
2	X	X	X	X	X	X		Doc	FIDIV/FIDIVR m16int description
3	X	X	X	X	X	X		Doc	PUSH does not pad with zeros
4	X	X	X	X	X	X		Doc	DR7, bit 10 is reserved
5	X	X	X	X	X	X		Doc	Additional states that are not automatically saved and restored
6	X	X	X	X				Doc	S.E.C. cartridge mechanical specification corrections
7	X	X	X	X	X	X		Doc	Cache and TLB description correction
8	X	X	X	X	X	X		Doc	SMRAM state save map contains documentation error

NO.	C0	C1	dA0	dA1	mdA0	mmdA0	SUB	Plans	SPECIFICATION CLARIFICATIONS
1	X	X	X	X	X	X		Doc	Writes to WC memory
2	X	X	X	X	X	X		Doc	Multiple processors protocol and restrictions
3	X	X	X	X	X	X		Doc	NMI handling while in SMM
4	X	X	X	X	X	X		Doc	Critical sequence of events during a page fault exception
5	X	X	X	X	X	X		Doc	Performance-monitoring counter issues
6	X	X	X	X	X	X		Doc	POP[ESP] with 16-bit stack size

NO.	C0	C1	dA0	dA1	mdA0	mmdA0	SUB	Plans	SPECIFICATION CHANGES
1	X	X	X	X				Doc	Mixing steppings in MP systems
2	X	X	X	X				Doc	System bus timings changes
3	X	X	X	X				Doc	FRCERR pin removed from specification

ERRATA

1. FP Data Operand Pointer May Be Incorrectly Calculated After FP Access Which Wraps 64-Kbyte Boundary in 16-Bit Code

PROBLEM: The FP Data Operand Pointer is the effective address of the operand associated with the last noncontrol floating-point instruction executed by the machine. If an 80-bit floating-point access (load or store) occurs in a 16-bit mode other than protected mode (in which case the access will produce a segment limit violation), the memory access wraps a 64-Kbyte boundary, and the floating-point environment is subsequently saved in 32-bit mode, the subtraction routine used to calculate the FP Data Operand Pointer will assume the floating-point access was in 32-bit mode, and the high word of the address will be FFFFh instead of 0000h.

IMPLICATION: A 32-bit operating system running 16-bit floating-point code may encounter this erratum, under the following conditions:

- The operating system is using a segment greater than 64 Kbytes in size.
- An application is running in a 16-bit mode other than protected mode.
- An 80-bit floating-point load which wraps the 64-Kbyte boundary is executed.
- The operating system uses a 32-bit handler on an unmasked exception which occurs during the load.
- The exception handler uses the value contained in the FP Data Operand Pointer.

Wrapping an 80-bit floating-point load around a segment boundary in this way is not a normal programming practice. Intel has not currently identified any software which exhibits this behavior.

WORKAROUND: If the FP Data Operand Pointer is used in a 32-bit exception handler in an OS which may run 16-bit floating-point code, care must be taken to ensure that no 80-bit floating-point accesses are wrapped around a 64-Kbyte boundary.

STATUS: For the steppings affected see the Summary Table of Changes at the beginning of this section.

2. Differences Exist in Debug Exception Reporting

PROBLEM: There exist some differences in the reporting of code and data breakpoint matches between that specified by previous Intel processors' specifications and the behavior of the Pentium II processor, as described below:

CASE 1:
The first case is for a breakpoint set on a MOVSS or POPSS instruction, when the instruction following it causes a debug register protection fault (DR7.gd is already set, enabling the fault). The Pentium processor reports delayed data breakpoint matches from the MOVSS or POPSS instructions by setting the matching DR6.bi bits, along with the debug register protection fault (DR6.bd). If additional breakpoint faults are matched during the call of the debug fault handler, the Pentium processor sets the breakpoint match bits (DR6.bi) to reflect the breakpoints matched by both the MOVSS or POPSS breakpoint and the debug fault handler call. The Pentium II processor only sets DR6.bd in either situation, and does not set any of the DR6.bi bits.

CASE 2:
In the second breakpoint reporting failure case, if a MOVSS or POPSS instruction with a data breakpoint is followed by a store to memory which crosses a 4-Kbyte page boundary, the breakpoint information for the MOVSS or POPSS will be lost. Previous processors retain this information across such a page split.

CASE 3:
If they occur after a MOVSS or POPSS instruction, the INT *n*, INTO, and INT3 instructions zero the DR6.Bi bits (bits B0 through B3), clearing pending breakpoint information, unlike previous processors.

CASE 4:
If a data breakpoint and an SMI (System Management Interrupt) occur simultaneously, the SMI will be serviced via a call to the SMM handler, and the pending breakpoint will be lost.

CASE 5:
When an instruction which accesses a debug register is executed, and a breakpoint is encountered on the instruction, the breakpoint is reported twice.

IMPLICATION: When debugging or when developing debuggers for a Pentium II processor-based system, this behavior should be noted. Normal usage of the MOVSS or POPSS instructions (i.e., following them with a MOV ESP) will not exhibit the behavior of cases 1-3. Debugging in conjunction with SMM will be limited by case 4.
WORKAROUND: Following MOVSS and POPSS instructions with a MOV ESP instruction when using breakpoints will avoid the first three cases of this erratum. No workaround has been identified for cases 4 or 5.

STATUS: For the steppings affected see the Summary Table of Changes at the beginning of this section.

3. FLUSH# Servicing Delayed While Waiting for STARTUP_IPI in 2-way MP Systems

PROBLEM: In a 2-way MP system, if an application processor is waiting for a startup interprocessor interrupt (STARTUP_IPI), then it will not service a FLUSH# pin assertion until it has received the STARTUP_IPI.

IMPLICATION: After the 2-way MP initialization protocol, only one processor becomes the bootstrap processor (BSP). The other processor becomes a slave application processor (AP). After losing the BSP arbitration, the AP goes into a wait loop, waiting for a STARTUP_IPI.

The BSP can wake up the AP to perform some tasks with a STARTUP_IPI, and then put it back to sleep with an initialization interprocessor interrupt (INIT_IPI, which has the same affect as asserting INIT#), which returns it to a wait loop. The result is a possible loss of cache coherency if the offline processor is intended to service a FLUSH# assertion at this point. The FLUSH# will be serviced as soon as the processor is awakened by a STARTUP_IPI, before any other instructions are executed. Intel has not encountered any operating systems that are affected by this erratum.

WORKAROUND: Operating system developers should take care to execute a WBINVD instruction before the AP is taken offline using an INIT_IPI.

STATUS: For the steppings affected see the Summary Table of Changes at the beginning of this section.

4. Code Fetch Matching Disabled Debug Register May Cause Debug Exception

PROBLEM: The bits L0-3 and G0-3 enable breakpoints local to a task and global to all tasks, respectively. If one of these bits is set, a breakpoint is enabled, corresponding to the addresses in the debug registers DR0-DR3. If at least one of these breakpoints is enabled, any of these registers are **disabled** (i.e., L*n* and G*n* are 0), and RW*n* for the disabled register is 00 (indicating a breakpoint on instruction execution), normally an instruction fetch will not cause an instruction-breakpoint fault based on a match with the address in the disabled register (s). However, if the address in a disabled register matches the address of a code fetch which also results in a page fault, an instruction-breakpoint fault will occur.

IMPLICATION: While debugging software, extraneous instruction-breakpoint faults may be encountered if breakpoint registers are not cleared when they are disabled. Debug software which does not implement a code breakpoint handler will fail, if this occurs. If

a handler is present, the fault will be serviced. Mixing data and code may exacerbate this problem by allowing disabled data breakpoint registers to break on an instruction fetch.

WORKAROUND: The debug handler should clear breakpoint registers before they become disabled.

STATUS: For the steppings affected see the Summary Table of Changes at the beginning of this section.

5. Double ECC Error on Read May Result in BINIT#

PROBLEM: For this erratum to occur, the following conditions must be met:

- Machine Check Exceptions (MCEs) must be enabled.
- A dataless transaction (such as a write invalidate) must be occurring simultaneously with a transaction which returns data (a normal read).
- The read data must contain a double-bit uncorrectable ECC error.

If these conditions are met, the Pentium II processor will not be able to determine which transaction was erroneous, and instead of generating an MCE, it will generate a BINIT#.

IMPLICATION: The bus will be reinitialized in this case. However, since a double-bit uncorrectable ECC error occurred on the read, the MCE handler (which is normally reached on a double-bit uncorrectable ECC error for a read) would most likely cause the same BINIT# event.

WORKAROUND: Though the ability to drive BINIT# can be disabled in the Pentium II processor, which would prevent the effects of this erratum, overall system behavior would not improve, since the error which would normally cause a BINIT# would instead cause the machine to shut down. No other workaround has been identified.

STATUS: For the steppings affected see the Summary Table of Changes at the beginning of this section.

6. FP Inexact-Result Exception Flag May Not Be Set

PROBLEM: When the result of a floating-point operation is not exactly representable in the destination format (1/3 in binary form, for example), an inexact-result (precision) exception occurs. When this occurs, the PE bit (bit 5 of the FPU status word) is normally set by the processor. Under certain rare conditions, this bit may not be set when this rounding occurs. However, other actions taken by the processor (invoking the software exception handler if the exception is unmasked) are not affected. This erratum

can only occur if the floating-point operation which causes the precision exception is immediately followed by one of the following instructions:

- FST m32real
- FST m64real
- FSTP m32real
- FSTP m64real
- FSTP m80real
- FIST m16int
- FIST m32int
- FISTP m16int
- FISTP m32int
- FISTP m64int

Note that even if this combination of instructions is encountered, there is also a dependency on the internal pipelining and execution state of both instructions in the processor.

IMPLICATION: Inexact-result exceptions are commonly masked or ignored by applications, as it happens frequently, and produces a rounded result acceptable to most applications. The PE bit of the FPU status word may not always be set upon receiving an inexact-result exception. Thus, if these exceptions are unmasked, a floating-point error exception handler may not recognize that a precision exception occurred. Note that this is a "sticky" bit, i.e., once set by an inexact-result condition, it remains set until cleared by software.

WORKAROUND: This condition can be avoided by inserting a NOP instruction between the two floating-point instructions.

STATUS: For the steppings affected see the Summary Table of Changes at the beginning of this section.

7. BTM for SMI Will Contain Incorrect FROM EIP

PROBLEM: A system management interrupt (SMI) will produce a Branch Trace Message (BTM), if BTMs are enabled. However, the FROM EIP field of the BTM (used to determine the address of the instruction which was being executed when the SMI was serviced) will not have been updated for the SMI, so the field will report the same FROM EIP as the previous BTM.

IMPLICATION: A BTM which is issued for an SMI will not contain the correct FROM EIP, limiting the usefulness of BTMs for debugging software in conjunction with System Management Mode (SMM).

WORKAROUND: None identified at this time.
STATUS: For the steppings affected see the Summary Table of Changes at the beginning of this section.

8. I/O Restart in SMM May Fail After Simultaneous MCE

PROBLEM: If an I/O instruction (IN, INS, REP INS, OUT, OUTS, or REP OUTS) is being executed, and if the data for this instruction becomes corrupted, the Pentium II processor will signal a machine check exception (MCE). If the instruction is directed at a device which is powered down, the processor may also receive an assertion of SMI#. Since MCEs have higher priority, the processor will call the MCE handler, and the SMI# assertion will remain pending. However, upon attempting to execute the first instruction of the MCE handler, the SMI# will be recognized and the processor will attempt to execute the SMM handler. If the SMM handler is completed successfully, it will attempt to restart the I/O instruction, but will not have the correct machine state, due to the call to the MCE handler.
IMPLICATION: A simultaneous MCE and SMI# assertion may occur for one of the I/O instructions above. The SMM handler may attempt to restart such an I/O instruction, but will have corrupted state due to the MCE handler call, leading to failure of the restart and SHUTDOWN of the processor.
WORKAROUND: If a system implementation must support both SMM and MCEs, the first thing the SMM handler code (when an I/O restart is to be performed) should do is check for a pending MCE. If there is an MCE pending, the SMM handler should immediately exit via an RSM instruction and allow the machine check exception handler to execute. If there is not, the SMM handler may proceed with its normal operation.
STATUS: For the steppings affected see the Summary Table of Changes at the beginning of this section.

9. Branch Traps Do Not Function if BTMs Are Also Enabled

PROBLEM: If branch traps or branch trace messages (BTMs) are enabled alone, both function as expected. However, if both are enabled, only the BTMs will function, and the branch traps will be ignored.
IMPLICATION: The branch traps and branch trace message debugging features cannot be used together.
WORKAROUND: If branch trap functionality is desired, BTMs must be disabled.
STATUS: For the steppings affected see the Summary Table of Changes at the beginning of this section.

10. Checker BIST Failure in FRC Mode Not Signaled

PROBLEM: If a system is running in functional redundancy checking (FRC) mode, and the checker of the master-checker pair encounters a hard failure while running the built-in self test (BIST), the checker will tri-state all outputs without signaling an IERR#.
IMPLICATION: Assuming the master passes BIST successfully, it will continue execution unchecked, operating without functional redundancy. However, the necessary pull-up on the FRCERR pin will cause an FRCERR to be signaled. The operation of the master depends on the implementation of FRCERR.
WORKAROUND: For successful detection of BIST failure in the checker of an FRC pair, use the FRCERR signal, instead of IERR#.
STATUS: For the steppings affected see the Summary Table of Changes at the beginning of this section.

11. BINIT# Assertion Causes FRCERR Assertion in FRC Mode

PROBLEM: If a pair of Pentium II processors are running in functional redundancy checking (FRC) mode, and a catastrophic error condition causes BINIT# to be asserted, the checker in the master-checker pair will enter SHUTDOWN. The next bus transaction from the master will then result in the assertion of FRCERR.
IMPLICATION: Bus initialization via an assertion of BINIT# occurs as the result of a catastrophic error condition which precludes the continuing reliable execution of the system. Under normal circumstances, the master-checker pair would remain synchronized in the execution of the BINIT# handler. However, due to this erratum, an FRCERR will be signaled. System behavior then depends on the behavior of the system specific error recovery mechanisms.
WORKAROUND: None identified at this time.
STATUS: For the steppings affected see the Summary Table of Changes at the beginning of this section.

12. Machine Check Exception Handler May Not Always Execute Successfully

PROBLEM: An asynchronous machine check exception (MCE), such as a BINIT# event, which occurs during an access that splits a 4-Kbyte page boundary may leave some internal registers in an indeterminate state. Thus, MCE handler code may not always run successfully if an asynchronous MCE has occurred previously.
IMPLICATION: An MCE may not always result in the successful execution of the MCE handler. However, asynchronous MCEs usually occur upon detection of a catastrophic system condition that would also hang the processor. Leaving MCEs disabled will result in the condition which caused the asynchronous MCE instead causing the

processor to enter SHUTDOWN. Therefore, leaving MCEs disabled may not improve overall system behavior.
WORKAROUND: No workaround which would guarantee successful MCE handler execution under this condition has been identified.
STATUS: For the steppings affected see the Summary Table of Changes at the beginning of this section.

13. MCE Due to L2 Parity Error Gives L1 MCACOD.LL

PROBLEM: If a Cache Reply Parity (CRP) error, Cache Address Parity (CAP) error, or Cache Synchronous Error (CSER) occurs on an access to the Pentium II processor's L2 cache, the resulting Machine Check Architectural Error Code (MCACOD) will be logged with '01' in the LL field. This value indicates an L1 cache error; the value should be '10', indicating an L2 cache error. Note L2 ECC errors have the correct value of '10' logged.
IMPLICATION: An L2 cache access error, other than an ECC error, will be improperly logged as an L1 cache error in MCACOD.LL.
WORKAROUND: None identified at this time.
STATUS: For the steppings affected see the Summary Table of Changes at the beginning of this section.

14. LBER May Be Corrupted After Some Events

PROBLEM: The last branch record (LBR) and the last branch before exception record (LBER) can be used to determine the source and destination information for previous branches or exceptions. The LBR contains the source and destination addresses for the last branch or exception, and the LBER contains similar information for the last branch taken before the last exception. This information is typically used to determine the location of a branch which leads to execution of code which causes an exception. However, after a catastrophic bus condition which results in an assertion of BINIT# and the reinitialization of the buses, the value in the LBER may be corrupted. Also, after either a CALL which results in a fault or a software interrupt, the LBER and LBR will be updated to the same value, when the LBER should not have been updated.
IMPLICATION: The LBER and LBR registers are used only for debugging purposes. When this erratum occurs, the LBER will not contain reliable address information. The value of LBER should be used with caution when debugging branching code; if the values in the LBR and LBER are the same, then the LBER value is incorrect. Also, the value in the LBER should not be relied upon after a BINIT# event.
WORKAROUND: None identified at this time.

STATUS: For the steppings affected see the Summary Table of Changes at the beginning of this section.

15. BTMs May Be Corrupted During Simultaneous L1 Cache Line Replacement

PROBLEM: When Branch Trace Messages (BTMs) are enabled and such a message is generated, the BTM may be corrupted when issued to the bus by the L1 cache if a new line of data is brought into the L1 data cache simultaneously. Though the new line being stored in the L1 cache is stored correctly, and no corruption occurs in the data, the information in the BTM may be incorrect due to the internal collision of the data line and the BTM.

IMPLICATION: Although BTMs may not be entirely reliable due to this erratum, the conditions necessary for this boundary condition to occur have only been exhibited during focused simulation testing. Intel has currently not observed this erratum in a system level validation environment.

WORKAROUND: None identified at this time.

STATUS: For the steppings affected see the Summary Table of Changes at the beginning of this section.

16. System May Hang Due to Internal Protocol Violation

PROBLEM: Pentium II processor-based systems may hang due to an internal protocol violation. When a snoopable transaction is issued on the bus and the cache line being accessed is in the modified state, the processor must deliver to the system bus an updated copy of the cache line. When the processor attempts to deliver the most up to date copy via an implicit writeback, the data transfer transaction fails and the DBSY# signal remains asserted until the next RESET#. This causes the system to hang indefinitely. In order to encounter this erratum, the following sequence of events must occur:

1. A snoopable transaction (transaction 1) is issued on the system bus. The processor contains in its L1 and/or L2 caches the data for this line in the modified state.
2. Another snoopable transaction (transaction 2) is issued and the processor contains this line only in its L2 cache in the modified state. Both of these transactions can be issued by either the chipset, by the processor (in which case they are of the self-snoop type), by another processor (2-way MP systems), or any combination thereof.
3. A nonsnoopable transaction is then issued (transaction 3) for which address bits A15-A5 are the same as those in transaction 2.
4. Transaction 3 is followed by a snoopable transaction (transaction 4).

5. The completion of the data transfer phase of transaction 1 must line up with the snoop response phase of transaction 3. This data transfer phase of transaction 1 must occur after the ADS# of transaction 4 and line up with the completion of an internal cache transaction.
6. The internal cache transaction must miss the L2 targeting a line for eviction, but the internal cache transaction must be such that it has to be retried.

The result of this sequence of transactions causes the processor bus to lock up after delivering the data for transaction 1, but prior to delivering the data for transaction 2. Since this data is never delivered, DBSY# does not deassert and the system hangs.
IMPLICATION: The Pentium II processor may cause a system to hang if the above listed sequence of events occurs. This sequence is a necessary condition to hit the erratum, but multiple variations of this sequence which also cause this erratum are also possible. The probability of encountering this erratum increases with I/O queue depth greater than 4 and in 2-way MP systems.
WORKAROUND: None identified at this time.
STATUS: For the steppings affected see the Summary Table of Changes at the beginning of this section.

17. Livelock Condition May Cause System Hang

PROBLEM: A “livelock” situation could occur in 2-way MP Pentium II processor-based systems, when IOQ depth is set to 1, with a failure signature such that a processor arbitrates for the system bus but fails to drive out a transaction when it gains ownership of the bus. The processor then relinquishes bus ownership to another requester, but on rearbitration performs the same repetitive actions. This course of action continues until RESET# is asserted. The failure signature in 2-way MP systems is such that both processors require execution of an explicit writeback cycle and both processors request the bus for this transaction. However, when the time comes to drive out the writeback transaction, the internal request has been suspended due to an internal blocking condition. After the internal blocking condition has gone away the original writeback request is reasserted. However, by the time bus ownership has been regained, the blocking condition has recurred, thus suppressing the writeback request before the transaction can be driven out to the system bus.
The writeback which is waiting to go out on the system bus must be issued before the internal blocking condition can be removed. But the writeback can never be issued because of the recurring blocking condition. This causes an “infinite loop” situation to develop, and the processor essentially stops executing code.

IMPLICATION: This erratum was observed to occur when both processors are configured for IOQ depth = 1 in Intel commercial system testing.
WORKAROUND: None identified at this time.
STATUS: For the steppings affected see the Summary Table of Changes at the beginning of this section.

18. Mispredicted Branch May Cause Incorrect Tag Word on MMX™ Technology Instructions

PROBLEM: After any MMX™ technology instruction is executed, all of the FPU stack registers should be marked valid in the FPU tag word. If one or more of the first three instructions of a mispredicted branch are MMX technology instructions of the form "opcode reg, mem" not including MOVD and MOVQ, the FPU tag word is incorrectly modified. Some of the tag word bits may remain invalid. This tag word will remain incorrect until one of two events occur:

1. Any MMX™ technology instruction is executed four or more instructions after the branch target, or
2. An MMX technology instruction of the following type is executed:
   - Any MMX technology instruction of the form "opcode reg, reg"
   - MOVD
   - MOVQ
   - EMMS

The following are examples of code that will encounter this erratum.

Example 1:

```
        EMMS
        ...
        Jcc     target          ; mispredicted as not taken
        ...
target:
        PADDW   mm0, [edi]      ; Is an "reg, mem" format instruction
        FSTENV  env
```

In this example, the tag word stored in memory by FSTENV will be incorrect.

Example 2:

```
        EMMS
        ...
        Jcc     target          ; mispredicted as not taken
        ...
target:
        PADDW       mm0, [edi]
        FUCOMPP                 ; depends on tag word, also violates coding guideline
        against mixing
                                ; floating-point and MMX™ technology instructions
        FWAIT
```

In this example, the FUCOMPP instruction will cause a Numeric Invalid Operation Exception if the FPU stack fault exception is unmasked.

IMPLICATION: When writing code that mixes FP and MMX technology instructions where the target of a branch is an MMX technology instruction with a memory operand, the FPU tag word may be incorrect. Software that expects the FP stack register to be set to valid after an MMX technology instruction and utilizes this information may be affected.

If floating-point instructions are intermixed, the floating-point instructions may raise the floating-point stack exception. If this exception is unmasked, the application will receive an unexpected numeric exception. The result is application dependent. If the floating-point stack exception is masked, the floating-point instruction will compute with a indefinite operand instead of the register contents. In either case the result is application dependent. Applications that follow the Intel MMX Technology Coding Guidelines against intermixing floating-point and MMX technology code are not affected by this erratum.

If the floating-point tag word is saved immediately after an affected MMX technology instruction, an erroneous value will be stored. Program behavior is application dependent. This may also cause debuggers to temporarily display incorrect tag word contents.

WORKAROUND: All of the following must be applied to work around this erratum:

- Follow the Intel MMX™ technology guidelines in the *Intel Architecture Developer's Optimization Manual* for writing MMX technology programs. Specifically, do not intermix MMX technology instructions and floating-point instructions on a per instruction basis.

- If it is possible that some of the tag word bits may be invalid prior to a branch, avoid using MMX technology instructions of the form “opcode reg, mem”, except MOVD, MOVQ, within the first three instructions at the target of a branch.
- Use the FSAVE instruction to save all floating-point stack registers if at least one of the registers is valid during a context switch.
- Before a nonstandard transition from MMX technology code to floating-point code, execute a nonsusceptible MMX technology instruction such as MOVD eax, mm0.
- Floating-point instructions should not depend on MMX technology instructions to set the tag word bits to valid.

STATUS: For the steppings affected see the Summary Table of Changes at the beginning of this section.

19. Thermal Sensor/THERMTRIP# Does Not Work

PROBLEM: THERMTRIP# is a feature of the Pentium II processor which asserts when the core reaches a certain temperature during operation as specified in the *Pentium® II Processor at 233 MHz, 266 MHz, 300 MHz, and 333 MHz* datasheet. The Pentium II processor may assert THERMTRIP# at a temperature lower or higher than the specified trippoint of 135 °C for $T_{JUNCTION}$. When THERMTRIP# is asserted, the processor may shut down causing all execution to be halted.

IMPLICATION: When running the Pentium II processor, the Pentium II processor core may reach a temperature causing the processor to assert THERMTRIP# early. Once THERMTRIP# has been asserted, the processor may shut down due to this erratum. All execution after the SHUTDOWN will be halted. This erratum is only exhibited when T_{PLATE} is above the Maximum Specification of 75 °C (see the *Pentium® II Processor at 233 MHz, 266 MHz, 300 MHz, and 333 MHz* datasheet, Order Number 243335, for details on specifications).

WORKAROUND: Avoid operation of the Pentium II processor outside of thermal specifications defined by the *Pentium® II Processor at 233 MHz, 266 MHz, 300 MHz, and 333 MHz* datasheet. Do not monitor the THERMTRIP# pin (pin A15).

STATUS: For the steppings affected see the Summary Table of Changes at the beginning of this section.

20. Spurious Machine Check Exception Via IFU Data Parity Error

PROBLEM: The Pentium II processor can signal an unrecoverable Machine Check Exception (MCE) in the event that the Instruction Fetch Unit (IFU) detects a mismatch when verifying instruction parity. The execution of code which modifies the current

instruction sequence that may already be fetched into the processor can cause an instruction at a given address to appear differently depending on when it was fetched in time relative to its being modified. Thus, a speculatively prefetched instruction may have been modified such that it now differs from the copy of the same instruction resident in the instruction cache. This discrepancy (of one copy located in the speculative prefetch portion, and a different copy in the instruction cache) is sensed by the IFU. When the IFU detects that the instruction stream has been modified, it flushes the pipeline and attempts to restart the instruction stream. In the interim, the IFU recognizes the disparate instructions described above, and signals a data parity error. The data parity error is signaled as an MCE before the instruction stream has had a chance to restart. This MCE will cause an operating system that has enabled MCE to shut down. No incorrect code is executed by the processor in this situation (even if MCE is disabled). Note that this erratum occurs under a specific set of address dependencies and timing events.

IMPLICATION: Executing such a sequence by modifying code without proper synchronization may not always result in predictable program behavior. The processor's signaling of an MCE due to a data parity error in the IFU may then result in an unexpected system halt if the above conditions are met and MCEs are enabled.

WORKAROUND: It is possible for BIOS code to contain a workaround for this erratum.

STATUS: For the steppings affected see the Summary Table of Changes at the beginning of this section.

21. Loss of Inclusion in IFU Can Cause Machine Check Exception

PROBLEM: The Pentium II processor can signal an unrecoverable Machine Check Exception (MCE) as a consistency checking mechanism in the event that the Instruction Fetch Unit (IFU) detects differences in the consistency of code in instruction streaming buffers against code resident in the instruction cache, i.e., a loss of inclusion. When application code makes an operating system call, the processor transitions execution privilege levels. If the code for the OS call is not already resident in the level 1 cache, then the processor may prefetch code while identifying a cache line(s) for eventual eviction to make space for the new code. Upon return from the OS call, the processor continues execution of application code at the user level. The processor, due to deep speculation and branch prediction, may attempt to execute instructions from the previously prefetched kernel code starting by attempting to replace the victim line with kernel code in a buffer internal to the IFU. The IFU detects that the current application is insufficiently privileged to execute the kernel code and so, suppresses the eviction of the previously selected victim line. Despite having detected this condition, the IFU does replace this victim line with the kernel line. If the processor now attempts to restart

execution of the current application code by refetching the original victim line it no longer finds it in the instruction cache. The IFU detects this loss of inclusion, and signals this by generating a MCE. If MCEs are enabled, this event can cause an operating system to shut down. Note that this erratum occurs under a specific set of address dependencies and timing events.
IMPLICATION: The occurrence of all the conditions above can lead the IFU to signal a loss of inclusion by generating an MCE. If MCEs are enabled in the system, then the operating system may shut down upon noticing the MCE resulting in system failure. If MCEs are disabled, then unpredictable application behavior is theoretically possible, although current validation has shown execution to continue normally.
WORKAROUND: It is possible for BIOS code to contain a workaround for this erratum.
STATUS: For the steppings affected see the Summary Table of Changes at the beginning of this section.

22. Possible System Hang When Paging is Disabled and Reenabled from Uncached Memory

PROBLEM: If paging is disabled via the PG bit of CR0 and then later reenabled while executing code from a page marked uncachable by its Page Table Entry (PCD=1) but located in memory mapped as Write Back or Write Through by the processor MTRRs, the processor could internally enter a state resulting in a system hang.
IMPLICATION: Operating systems that enable and disable paging with the above described memory configurations could hang. Intel has not observed this erratum to date in laboratory testing of commercially available operating systems and applications.
WORKAROUND: It is possible for BIOS code to contain a workaround for this erratum.
STATUS: For the steppings affected see the Summary Table of Changes at the beginning of this section.

23. L2 Performance Counters Miscount L2_RQSTS

PROBLEM: L2_RQSTS is a performance counter that counts the number of L2 cache access requests. This counter increments for each incoming L2 cache request. In some cases, an L2 request cannot be serviced by the L2 Cache. This request is then retried at a later time when the request can be serviced by the L2 cache. When this happens, the L2_RQSTS counter counts the initial L2 cache request **and** the retried L2 cache request, thereby counting the same request twice.
IMPLICATION: The L2_RQSTS counter may contain a larger erroneous number of L2 cache requests due to this erratum. This erratum does not affect functionality of the Pentium II processor. This erratum only affects the performance counter specified.

WORKAROUND: None identified at this time.
STATUS: For the steppings affected see the Summary Table of Changes at the beginning of this section.

24. Erroneous Signaling of User Mode Protection Violation

PROBLEM: If the Pentium II processor attempts to access a page in physical memory marked not present (Present bit clear), a page fault exception (#PF) is generated. Before proceeding, there is a narrow internal timing window where the processor verifies that no other higher priority fault conditions are present. During this time, it is possible for another agent to allocate a new page directory or page table entry (PDE/PTE) corresponding to the same linear address of the original access, writing new values into the PDE/PTE with the Access bit (A-bit) or the Dirty bit (D-bit) cleared. When the original processor completes its checking for other fault conditions, and reexamines the A/D bit of the recently modified PDE/PTE, it finds that it has been cleared. Internal hardware correctly signals this scenario as a condition to which the processor should respond by setting the A/D bit, but erroneously reports it as a generic paging protection violation. Instead of attempting to set the appropriate A/D bit, this event is reported as an Int14 with exception code 0x05, i.e., user mode protection violation.
IMPLICATION: The occurrence of this scenario will result in the erroneous signaling of a user mode protection violation instead of a page fault and may result in application termination depending on operating system behavior in response to a user mode protection violation. Intel has only observed this erratum to date in laboratory testing of multi-processor systems.
WORKAROUND: Operating systems which allocate new PTEs and PDEs should set the Access bit (A-bit) and Dirty bit (D-bit) to work around this erratum. Alternately, an operating system's Int14 handler can determine if a protection violation condition truly exists, and if none is found, return without further action.
STATUS: For the steppings affected see the Summary Table of Changes at the beginning of this section.

25. Invalid Operation Not Signaled By the FIST Instruction on Some Out of Range Operands

PROBLEM: On certain, large, negative, floating-point operands, and only in three of the four possible processor rounding modes, the instructions FIST[P] m16int and FIST[P] m32int do not detect that the operand is so large that it will not fit into the target data size. As a consequence, the expected Invalid Operation exception response for this situation is not correctly provided, nor is the Invalid Operation flag set in the Floating-

Point status word as specified in the *Intel Architecture Programmers Reference Manual, Volume 2*. Under the failing conditions, noted below, the precision exception (#PE) will also be incorrectly set.
The erratum occurs only when all of the following conditions are met:

1. Operations are unaffected.
2. The FIST[P] instruction is either a 16- or 32-bit operation; 64-bit Either the '*to nearest*', '*to zero*' or '*up*' rounding modes are being used. The round '*down*' mode is unaffected by this erratum.
3. The sign bit of the floating-point operand is negative.
4. The floating-point operand being converted is significantly more negative than can be described by the integer size being targeted.

ACTUAL vs. EXPECTED RESPONSE
A. Actual Response:
When the required conditions are encountered, the processor provides the following response:

- Return the MAXNEG value (8000h for FIST16 & 80000000h for FIST32) to memory.
- The IE (Invalid Operation) bit in the Floating-Point status word is *not set* to flag the use of an invalid operand.
- The PE (precision error) bit the Floating-Point status word is *set*.
- No exception handler is invoked.
- In the case of a FISTP instruction the Operand will have been popped from the floating-point stack.

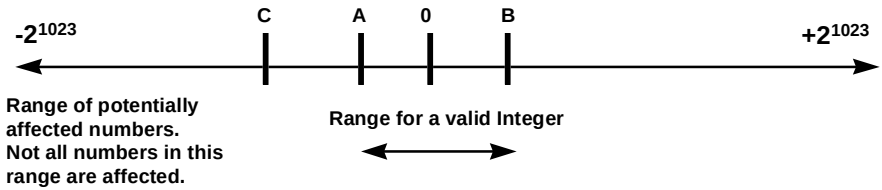
B. Expected Response
The expected processor response when the invalid operation exception is *masked* is:

- Return the MAXNEG value (8000h for FIST16 & 80000000h for FIST32) to memory.
- The IE (Invalid Operation) bit in the Floating-Point status word is *set* to flag the overflow.
- The PE (precision error) bit the Floating-Point status word is *not set*.

The expected processor response when the invalid operation exception is *unmasked* is:

- Do not return a result to memory. Keep the original operand intact on the stack.
- The IE (Invalid Operation) bit in the Floating-Point status word is *set* to flag the overflow.
- The PE (precision error) bit the Floating-Point status word is *not set*.
- Vector to the user numeric exception handler.

IMPLICATION: Erroneous operation results when the operand is so large that it will not fit into the target data size. The operands affected by this erratum are significantly outside (by a factor of 3X) the range that can be, correctly, converted to an integer value. The figure below and corresponding table identifies the normal range of integer numbers (between A and B) and the starting point of the operands affected by this erratum. Discrete failing operands will be present in the range between point C and the maximum negative number that can be represented by the processor (-2^{1023} in double precision format). Note 2 below gives a qualitative description of the nature of the discrete failing values. Software that does not rely on the Invalid Operation exception flag being set and signaled by either an exception OR by software polling is not impacted by this erratum.



16-bit Operation	A	B	C
	-32,768.0	+32,767.0	< -98304.0
32-bit Operation	A	B	C
	-2,147,483,648.0	+2,147,483,647.0	< -6,442,450,944.0

WORKAROUND: Any of two software workarounds will avoid occurrence of this erratum:

1. Range checking performed prior to execution of the FIST[P] instruction will prevent the overflow condition from occurring, and may already be implemented as a standard coding style.

2. Software can use the presence of MAXNEG in the result integer to indicate that an out of range conversion may have occurred.

Note 1: A possible alternative is to use the FIST64 instruction to store the converted operand to memory and access the lower 16 or 32 bits as the required integer. Even though this mechanism will not signal an attempted out of range conversion with a 16 bit or 32 bit target, it is currently in use by many compilers today.
Note 2: The values affected by this erratum are those which contain an exponent value within the affected range, AND a specific bit pattern at a specific offset within the mantissa, AND at least one nonzero bit to the right of the above bit pattern. The offset within the mantissa is a function of the floating-point exponent value. The specific bit pattern is 0x8000 for FIST16 and 0x80000000 for FIST32. This means that for any given exponent within the range, one mantissa value in every 2^{16} possible mantissa values exhibits the erratum for FIST16, and one mantissa value in every 2^{32} possible mantissa values exhibits the erratum for FIST32.

Examples of affected values for FIST16, 80 bit binary notation (not an exhaustive list) (xx means any bit pattern, yy means any nonzero bit pattern)

sign	exponent	mantissa
1	100000000010100	1xxxxx1000000000000000yyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyy
1	100000000010101	1xxxxxx1000000000000000yyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyy
1	100000000010110	1xxxxxxx1000000000000000yyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyy

Examples of affected values for FIST32, 80 bit binary notation (not an exhaustive list) (xx means any bit pattern, yy means any nonzero bit pattern)

sign	exponent	mantissa
1	100000000100011	1xxxx10000000000000000000000000000yyyyyyyyyyyyyyyyyyyyyyyyy
1	100000000100100	1xxxxx10000000000000000000000000000yyyyyyyyyyyyyyyyyyyyyyyyy
1	100000000100101	1xxxxxx10000000000000000000000000000yyyyyyyyyyyyyyyyyyyyyyyyy

STATUS: For the steppings affected see the Summary Table of Changes at the beginning of this section.

26. FLUSH# Assertion Disables L2 Machine Check Exception Reporting

PROBLEM: Upon FLUSH# assertion, the L2 Machine Check Exception generation is disabled. Once the FLUSH# pin is asserted, the processor disables the L2 MCA, by

clearing the associated MCi_CTL control register to “0”s. This operation is invisible to the software being executed.

IMPLICATION: Errors that should be reported by the L2 MCA are not reported from the time that the FLUSH# signal is asserted until the time that the MCi_CTL register is written back to all “1”s. All other errors will continue to be logged as normal.

WORKAROUND: Platform specific code (e.g., BIOS or system management software) has the potential for driving a device to assert the FLUSH# pin. If the platform specific code asserts the FLUSH# pin, this code should be enhanced to detect that MCA Exceptions are globally enabled (via register CR4.MCE). The code should then write “0”s to all of the MCi_CTL registers to clear any spurious entries and then write “1”s to all of the MCi_CTL registers in order to reenable exception reporting. Hardware devices in systems that require L2 error reporting which could assert the FLUSH# pin should not assert FLUSH#.

STATUS: For the steppings affected see the Summary Table of Changes at the beginning of this section

27. EFLAGS May Be Incorrect After a Multiprocessor TLB Shootdown

PROBLEM: When the Pentium II processor executes a read-modify-write arithmetic instruction with memory as the destination, it is possible for a page fault to occur during the execution of the store on the memory operand after the read operation has completed but before the write operation completes. In this case the EFLAGS value pushed onto the stack of the page fault handler may be reflective of the status of the EFLAGS register after the instruction would have completed execution rather than that before it has executed under a certain set of circumstances. This class of instruction will initially perform a load operation that has the side effect of ensuring that the final store portion of the instruction will successfully complete. The load ensures this by bringing the page table information of the page containing the data into the DTLB. This page entry could be evicted from the DTLB by speculative loads from other instructions that hit the same way of the DTLB, before the store is executed. DTLB eviction will require at least three load operations that have linear address bits 15:12 equal to each other and address bits 31:16 different from each other in close physical proximity to the arithmetic operation. If, in the very small window of time between the page eviction and the store execution, the page table entry has had its page permissions tightened (e.g., from Present to Not Present, or from Read/Write to Read Only, etc.) by the operating system in main memory by another processor (with no corresponding synchronization and subsequent TLB flush, the store will generate a DTLB miss and a call to the OS’s page fault handler. The EFLAGS register may have already been updated by the arithmetic portion of the instruction before entry to the page fault

handler. If under these circumstances the fault handler elects to restart the instruction the reexecution may generate an incorrect result. Instructions affected by this erratum are the memory destination forms of ADC, SBB, RCR & RCL (instructions that use a flag, carry, as input to the instruction). It should be noted that the locked version of these instructions is not impacted by this erratum.

IMPLICATION: This scenario can only occur in a multiprocessor system running under an operating system that implements a "lazy" TLB shootdown. Lazy TLB shootdown occurs when one processor makes changes to the page tables in memory, and then signals other processors to remove the page entry from their TLB without a multiprocessor synchronization being performed. To date, Intel has not observed this erratum in any laboratory testing of commercially available software applications.

In a multiprocessor system the arithmetic flags of the EFLAGS register and its memory stack image, may contain incorrect data if the read-modify-write arithmetic instruction encounters a page fault. Page Fault handler software that uses the resulting EFLAGS may see incorrect information. If the original instruction is restarted by the page fault handler, the instruction may produce incorrect results based on the prior modifications of the EFLAGS register.

WORKAROUND: Software may use the locked form of the ADC, SBB, RCR & RCL instructions to avoid this erratum. Operating systems should ensure that no processor is currently accessing a page that is scheduled to have its page permissions tightened, e.g., moved from Present to Not Present or have a page fault handler that can handle any incorrect state. Intel is working with Multiprocessor Operating System vendors to ensure that an OS level workaround is implemented as required.

STATUS: For the steppings affected see the Summary Table of Changes at the beginning of this section.

28. Delayed Line Invalidation Issue During 2-Way MP Data Ownership Transfer

PROBLEM: In 2-way MP systems, each processor may attempt to modify a different portion of the same cache line, referenced as line 'A' in the discussion below. When this erratum occurs (with the following example given for a 2-way MP system with processors noted as 'P0' and 'P1'), each processor contains a shared copy of line A in both their L1 and L2 caches. Each processor must issue an invalidation cycle before that processor can definitively source the results of its internal write to a portion of line A to the other processors.

There exists a narrow timing window when, if P0 wins the external bus invalidation race and gains ownership rights to line A due to the sequence of bus invalidation traffic, P1 may not have completed the pending invalidation of its own, currently valid and shared copy of line A. During this window, it is possible for a P1 internal opportunistic

write to a portion of line A (while awaiting ownership rights) to occur with the original shared copy of line A still resident in P1's L2 cache. Such internal modification is permissible subject to delaying the broadcast of such changes until line ownership has actually been gained. However, the processor must ensure that any internal reread by P1 of line A returns with data in the order actually written; in this case, this should be the data written by P0. In the case of this erratum, the internal reread uses the data which was written by P1.

IMPLICATION: Multiprocessor or threaded application synchronization that is implemented via operating system-provided synchronization constructs are not affected by this erratum. Applications which rely upon the usage of locked semaphores rather than memory ordering are also unaffected. Uniprocessor systems are not affected by this erratum. Intel has not identified, to date, any commercially available application or operating system software which is affected by this erratum. If the erratum does occur, the delayed line invalidation that occurs naturally due to the fact that one processor will necessarily win the invalidation race allows a narrow timing window to exist where one processor may reread a line that it just wrote internally, but return with the stale data that was present from the previous shared state rather than the data written more recently by another processor.

WORKAROUND: Deterministic barriers beyond which program variables will not be modified can be achieved via the usage of locked semaphore operations, and this scheme has been shown to effectively work around this erratum.

STATUS: For the steppings affected see the Summary Table of Changes at the beginning of this section.

29. Potential Early Deassertion of LOCK# During Split-Lock Cycles

PROBLEM: During a split-lock cycle there are four bus transactions: 1st ADS# (a partial read), 2nd ADS# (a partial read), 3rd ADS# (a partial write), and the 4th ADS# (a partial write). Due to this erratum, LOCK# may deassert one clock after the 4th ADS# of the split-lock cycle instead of after the RS# assertion corresponding to the 4th ADS# has been sampled. The following sequence of events are required for this erratum to occur:

1. A lock cycle occurs (split or nonsplit).
2. Five more bus transactions (assertion of ADS#) occur.
3. A split-lock cycle occurs and BNR# toggles after the 3rd ADS# (partial write) of the split-lock cycle. This in turn delays the assertion of the 4th ADS# of the split-lock cycle. BNR# toggling at this time could most likely happen when the bus is set for an IOQ depth of 2.

When all of these events occur, LOCK# will be deasserted in the next clock after the 4th ADS# of the split-lock cycle.
IMPLICATION: This may affect chipset logic which monitors the behavior of LOCK# deassertion.
WORKAROUND: None identified.
STATUS: For the steppings affected see the Summary Table of Changes at the beginning of this section.

30. A20M# May Be Inverted After Returning From SMM and Reset

PROBLEM: This erratum is seen when software causes the following events to occur:

1. The assertion of A20M# in real address mode.
2. After entering the 1-Mbyte address wrap-around mode caused by the assertion of A20M#, there is an assertion of SMI# intended to cause a Reset or remove power to the processor. Once in the SMM handler, software saves the SMM state save map to an area of nonvolatile memory from which it can be restored at some point in the future. Then software asserts RESET# or removes power to the processor.
3. After exiting Reset or completion of power-on, software asserts SMI# again. Once in the SMM handler, it then retrieves the old SMM state save map which was saved in event 2 above and copies it into the current SMM state save map. Software then asserts A20M# and executes the RSM instruction. After exiting the SMM handler, the polarity of A20M# is inverted.

IMPLICATION: If this erratum occurs, A20M# will behave with a polarity opposite from what is expected (i.e., the 1-Mbyte address wrap-around mode is enabled when A20M# is deasserted, and does not occur when A20M# is asserted).
WORKAROUND: Software should save the A20M# signal state in nonvolatile memory before an assertion of RESET# or a power down condition. After coming out of Reset or at power on, SMI# should be asserted again. During the restoration of the old SMM state save map described in event 3 above, the entire map should be restored, except for bit 5 of the byte at offset 7F18h. This bit should retain the value assigned to it when the SMM state save map was created in event 3. The SMM handler should then restore the original value of the A20M# signal.
STATUS: For the steppings affected see the Summary Table of Changes at the beginning of this section.

31. Reporting of Floating-Point Exception May Be Delayed

PROBLEM: The Pentium II processor normally reports a floating-point exception for an instruction when the next floating-point or MMX technology instruction is executed. The assertion of FERR# and/or the INT 16 interrupt corresponding to the exception may be delayed until the floating-point or MMX technology instruction **after** the one which is expected to trigger the exception, if the following conditions are met:

1. A floating-point instruction causes an exception.
2. Before another floating-point or MMX™ technology instruction, any one of the following occurs:
   a. A subsequent data access occurs to a page which has not been marked as accessed, or
   b. Data is referenced which crosses a page boundary, or
   c. A possible page-fault condition is detected which, when resolved, completes without faulting.
3. The instruction causing event 2 above is followed by a MOVQ or MOVD store instruction.

IMPLICATION: This erratum only affects software which operates with floating-point exceptions unmasked. Software which requires floating-point exceptions to be visible on the next floating-point or MMX technology instruction, and which uses floating-point calculations on data which is then used for MMX technology instructions, may see a delay in the reporting of a floating-point instruction exception in some cases. Note that mixing floating-point and MMX technology instructions in this way is not recommended.

WORKAROUND: Inserting a WAIT or FWAIT instruction (or reading the floating-point status register) between the floating-point instruction and the MOVQ or MOVD instruction will give the expected results. This is already the recommended practice for software.

STATUS: For the steppings affected see the Summary Table of Changes at the beginning of this section.

32. EFLAGS Discrepancy on a Page Fault After a Multiprocessor TLB Shootdown

PROBLEM: This erratum may occur when the Pentium II processor executes one of the following read-modify-write arithmetic instructions and a page fault occurs during the store of the memory operand: ADD, AND, BTC, BTR, BTS, CMPXCHG, DEC, INC, NEG, NOT, OR, ROL/ROR, SAL/SAR/SHL/SHR, SHLD, SHRD, SUB, XOR, and

XADD. In this case, the EFLAGS value pushed onto the stack of the page fault handler may reflect the status of the register after the instruction would have completed execution rather than before it. The following conditions are required for the store to generate a page fault and call the operating system page fault handler:

1. The store address entry must be evicted from the DTLB by speculative loads from other instructions that hit the same way of the DTLB before the store has completed. DTLB eviction requires at least three load operations that have linear address bits 15:12 equal to each other and address bits 31:16 different from each other in close physical proximity to the arithmetic operation.
2. The page table entry for the store address must have its permissions tightened during the very small window of time between the DTLB eviction and execution of the store. Examples of page permission tightening include from Present to Not Present or from Read/Write to Read Only, etc.
3. Another processor, without corresponding synchronization and TLB flush, must cause the permission change.

IMPLICATION: This scenario may only occur on a multiprocessor platform running an operating system that performs "lazy" TLB shootdowns. The memory image of the EFLAGS register on the page fault handler's stack prematurely contains the final arithmetic flag values although the instruction has not yet completed. Intel has not identified any operating systems that inspect the arithmetic portion of the EFLAGS register during a page fault nor observed this erratum in laboratory testing of software applications.

WORKAROUND: No workaround is needed upon normal restart of the instruction, since this erratum is transparent to the faulting code and results in correct instruction behavior. Operating systems may ensure that no processor is currently accessing a page that is scheduled to have its page permissions tightened or have a page fault handler that ignores any incorrect state.

STATUS: For the steppings affected see the Summary Table of Changes at the beginning of this section.

33. Near CALL to ESP Creates Unexpected EIP Address

PROBLEM: As documented, the CALL instruction saves procedure linking information in the procedure stack and jumps to the called procedure specified with the destination (target) operand. The target operand specifies the address of the first instruction in the called procedure. This operand can be an immediate value, a general purpose register, or a memory location. When accessing an absolute address indirectly using the stack pointer (ESP) as a base register, the base value used is the value in the ESP register

before the instruction executes. However, when accessing an absolute address directly using ESP as the base register, the base value used is the value of ESP **after** the return value is pushed on the stack, not the value in the ESP register **before** the instruction executed.

IMPLICATION: Due to this erratum, the processor may transfer control to an unintended address. Results are unpredictable, depending on the particular application, and can range from no effect to the unexpected termination of the application due to an exception. Intel has observed this erratum only in a focused testing environment. Intel has not observed any commercially available operating system, application, or compiler that makes use of or generates this instruction.

WORKAROUND: If the other seven general purpose registers are unavailable for use, and it is necessary to do a CALL via the ESP register, first push ESP onto the stack, then perform an **indirect** call using ESP (e.g., CALL [ESP]). The saved version of ESP should be popped off the stack after the call returns.

STATUS: For the steppings affected see the Summary Table of Changes at the beginning of this section.

34. Deep Sleep Exit Transition May Cause Hang

PROBLEM: Under normal operating conditions, when a system enters a power conservation mode, it enters System Management Mode (SMM), puts the processor in the Stop Grant State, followed by Sleep State and then may enter Deep Sleep State. Upon a resume event, the processor exits Deep Sleep but remains in SMM execution space until the SMI handler completes the system resume cycle.

If, prior to entering the Deep Sleep, the system was in SMM space, it is possible for the processor to exit Deep Sleep state and begin making accesses in the ‘normal’ memory space instead of staying in SMM space. The converse is also possible, i.e., if the processor is in ‘normal’ space prior to entering the Deep Sleep state, the processor may exit Deep Sleep and make accesses in SMM space instead.

IMPLICATION: Systems may execute incorrect code after exiting Deep Sleep, due to accesses to incorrect address space. This may produce unpredictable behavior, most likely hanging the system.

WORKAROUND: Avoid entering Deep Sleep. The table below offers the possible state transitions:

#	System State		Processor State		Possible Solutions	
	Name	ACPI Equivalent	Transition	ACPI Equivalent	Description	Suggested Solution
1	Active	S0	Normal to Stop Grant	C0, C1	N/A	None necessary
2	Active	S0	Stop Grant to Sleep to Deep Sleep	C1, C3	Use Stop Grant/Sleep only; do not use Deep Sleep	BIOS can specify the C3 latency time to be >1000 µs in the ACPI FACP table (P_LVL3_LAT, worst case hardware latency for the C3 state).
3	Powered On Suspend	S1, S2	Stop Grant to Deep Sleep	C1, C3	Reset CPU only and flush the cache without resetting the PCI bus, i.e., use POS_CCL state (POS with CPU Context Lost) instead of POS state.	BIOS can prevent the OS from entering the S1 state by NOT defining the S1 object in the ACPI DSDT table. Ensure that the cache is always flushed.

STATUS: For the steppings affected see the Summary Table of Changes at the beginning of this section.

35. Built-in Self Test Always Gives Nonzero Result

PROBLEM: The Built-in Self Test (BIST) of the Pentium II processor does not give a zero result to indicate a passing test. Regardless of pass or fail status, bit 6 of the BIST result in the EAX register after running BIST is set.

IMPLICATION: Software which relies on a zero result to indicate a passing BIST will indicate BIST failure.
WORKAROUND: Mask bit 6 of the BIST result register when analyzing BIST results.
STATUS: For the steppings affected see the Summary Table of Changes at the beginning of this section.

36. THERMTRIP# May Not Be Asserted as Specified

PROBLEM: THERMTRIP# is a signal on the Pentium II processor which is asserted when the core reaches a critical temperature during operation as detailed in the processor specification. The Pentium II processor may not assert THERMTRIP# until a much higher temperature than the one specified is reached.
IMPLICATION: The THERMTRIP# feature is not functional on the Pentium II processor. Note that this erratum can only occur when the processor is running with a T_{PLATE} temperature over the maximum specification of 75 °C.
WORKAROUND: Avoid operation of the Pentium II processor outside of the thermal specifications defined by the processor specifications.
STATUS: For the steppings affected see the Summary Table of Changes at the beginning of this section.

37. Cache State Corruption in the Presence of Page A/D-bit Setting and Snoop Traffic

PROBLEM: If an operating system uses the Page Access and/or Dirty bit feature implemented in the Intel architecture and there is a significant amount of snoop traffic on the bus, while the processor is setting the Access and/or Dirty bit the processor may inappropriately change a single L1 cache line to the modified state.
IMPLICATION: The occurrence of this erratum may result in cache incoherency, which may cause parity errors, data corruption (with no parity error), unexpected application or operating system termination, or system hangs.
WORKAROUND: It is possible for BIOS code to contain a workaround for this erratum.
STATUS: For the steppings affected see the Summary Table of Changes at the beginning of this section.

38. Snoop Cycle Generates Spurious Machine Check Exception

PROBLEM: The processor may incorrectly generate a Machine Check Exception (MCE) when it processes a snoop access that does not hit the L1 data cache. Due to an internal logic error, this type of snoop cycle may still check data parity on undriven data lines.

The processor generates a spurious machine check exception as a result of this unnecessary parity check.
IMPLICATION: A spurious machine check exception may result in an unexpected system halt if Machine Check Exception reporting is enabled in the operating system.
WORKAROUND: It is possible for BIOS code to contain a workaround for this erratum.
STATUS: For the steppings affected see the Summary Table of Changes at the beginning of this section.

39. MOVD/MOVQ Instruction Writes to Memory Prematurely

PROBLEM: When an instruction encounters a fault, the faulting instruction should not modify any CPU or system state. However, when the MMX technology store instructions MOVD and MOVQ encounter any of the following events, it is possible for the store to be committed to memory even though it should be canceled:

1. If CR0.EM = 1 (Emulation bit), then the store could happen prior to the triggered invalid opcode exception.
2. If the floating-point Top-of-Stack (FP TOS) is not zero, then the store could happen prior to executing the processor assist routine that sets the FP TOS to zero.
3. If there is an unmasked floating-point exception pending, then the store could happen prior to the triggered unmasked floating-point exception.
4. If CR0.TS = 1 (Task Switched bit), then the store could happen prior to the triggered Device Not Available (DNA) exception.

If the MOVD/MOVQ instruction is restarted after handling any of the above events, then the store will be performed again, overwriting with the expected data. The instruction will not be restarted after event 1. The instruction will definitely be restarted after events 2 and 4. The instruction may or may not be restarted after event 3, depending on the specific exception handler.
IMPLICATION: This erratum causes unpredictable behavior in an application if MOVD/MOVQ instructions are used to manipulate semaphores for multiprocessor synchronization, or if these MMX instructions are used to write to uncacheable memory or memory mapped I/O that has side effects, e.g., graphics devices. This erratum is completely transparent to all applications that do not have these characteristics. When each of the above conditions are analyzed:

1. Setting the CR0.EM bit forces all floating-point/MMX™ instructions to be handled by software emulation. The MOVD/MOVQ instruction, which is an MMX instruction, would be considered an invalid instruction. Operating systems typically terminates the application after getting the expected invalid opcode fault.

2. The FP TOS not equal to 0 case only occurs when the MOVD/MOVQ store is the first MMX instruction in an MMX technology routine and the previous floating-point routine did not clean up the floating-point states properly when it exited. Floating-point routines commonly leave TOS to 0 prior to exiting. For a store to be executed as the first MMX instruction in an MMX technology routine following a floating-point routine, the software would be implementing instruction level intermixing of floating-point and MMX instructions. Intel does not recommend this practice.
3. The unmasked floating-point exception case only occurs if the store is the first MMX technology instruction in an MMX technology routine and the previous floating-point routine exited with an unmasked floating-point exception pending. Again, for a store to be executed as the first MMX instruction in an MMX technology routine following a floating-point routine, the software would be implementing instruction level intermixing of floating-point and MMX instructions. Intel does not recommend this practice.
4. Device Not Available (DNA) exceptions occur naturally when a task switch is made between two tasks that use either floating-point instructions and/or MMX instructions. For this erratum, in the event of the DNA exception, data from the prior task may be temporarily stored to the present task's program state.

WORKAROUND: Do not use MMX instructions to manipulate semaphores for multiprocessor synchronization. Do not use MOVD/MOVQ instructions to write directly to I/O devices if doing so triggers user visible side effects. An OS can prevent old data from being stored to a new task's program state by cleansing the FPU explicitly after every task switch. Follow Intel's recommended programming paradigms in the *Intel Architecture Developer's Optimization Manual* for writing MMX technology programs. Specifically, do not mix floating-point and MMX instructions. When transitioning to new a MMX technology routine, begin with an instruction that does not depend on the prior state of either the MMX technology registers or the floating-point registers, such as a load or PXOR mm0, mm0. Be sure that the FP TOS is clear before using MMX instructions.

STATUS: For the steppings affected see the Summary Table of Changes at the beginning of this section.

40. Memory Type Undefined for Nonmemory Operations

PROBLEM: The Memory Type field for nonmemory transactions such as I/O and Special Cycles are undefined. Although the Memory Type attribute for nonmemory operations

logically should (and usually does) manifest itself as UC, this feature is not designed into the implementation and is therefore inconsistent.
IMPLICATION: Bus agents may decode a non-UC memory type for nonmemory bus transactions.
WORKAROUND: Bus agents must consider transaction type to determine the validity of the Memory Type field for a transaction.
STATUS: For the steppings affected, see the Summary Table of Changes at the beginning of this section.

41. Infinite Snoop Stall During L2 Initialization of MP Systems

PROBLEM: It is possible for snoop traffic generated on the system bus while a processor is executing its L2 cache initialization routine to cause the initializing processor to hang.
IMPLICATION: A 2-way MP system which does not suppress snoop traffic while L2 caches are being initialized may hang during this initialization sequence.
WORKAROUND: System BIOS can create an execution environment which allows processors to initialize their L2 caches without the system generating any snoop traffic on the bus.
Below is a pseudo-code fragment, designed explicitly for a 4 processor system, that uses a serial algorithm to initialize each processor's L2 cache:

```
Suppress_all_I/O_traffic()
 K = 0;
 while (K <= 3)
 {
 /* Obtain current value of K. This forces both Temp and K into */
 /* the L1 cache. Note that Temp could also be maintained in a */
 /* general purpose register. */

 Temp = K;
 Wait_until_all_processors_are_signed_in_at_barrier()
 if ( logical_proc_APIC_id == K ) {
 {
 wait_10_usecs_delay_loop(); /* this time delay, required */
 /* in the worst case, allows */
 /* the barrier semaphore to */
 /* settle to shared state. */
 Initialize L2 cache
 K++
 }
 else
 while (Temp == K);
 }
 }
```

This algorithm prevents bus snoop traffic from the other processors, which would otherwise cause the initializing processor to hang. The algorithm assumes that the L1 cache is enabled (the Temp and K variables must be cached by each processor). Also, the Memory Type Range Register (MTRR) for the data segment must be set to WB (writeback) memory type.

STATUS: For the steppings affected see the Summary Table of Changes at the beginning of this section.

42. Bus Protocol Conflict with Optimized Chipsets

PROBLEM: A "dead" turnaround cycle with no agent driving the address, address parity, request command, or request parity signals must occur between the processor driving these signals and the chipset driving them after asserting BPRI#. The Pentium II processor does not follow this protocol. Thus, if a system uses a chipset or third party agent which optimizes its arbitration latency (reducing it to 2 clocks when it observes an active (low) ADS# signal and an inactive (high) LOCK# signal on the same clock that BPRI# is asserted (driven low)), the Pentium II processor may cause bus contention during an unlocked bus exchange.

IMPLICATION: This violation of the bus exchange protocol when using a reduced arbitration latency may cause a system-level setup timing violation on the address, address parity, request command, or request parity signals on the system bus. This may result in a system hang or assertion of the AERR# signal, causing an attempted corrective action or shutdown of the system, as the system hardware and software dictate. The possibility of failure due to the contention caused by this erratum may be increased due to the processor's internal active pull-up of these signals on the clock after the signals are no longer being driven by the processor.

WORKAROUND: If the chipset and third party agents used with the Pentium II processor do not optimize their arbitration latency as described above, no action is required. For the 66 MHz Pentium II processor, no action is required. If agents that have implemented this optimization, by following the *100 MHz GTL+ Layout Guidelines for the Pentium® II Processor and Intel 440BX AGPset* for UP-SET (single-ended termination) and DP with both processors installed, no action is required. The following two cases are additional guidelines for UP-DET (dual-ended termination) and DP-DET with only one processor populated.

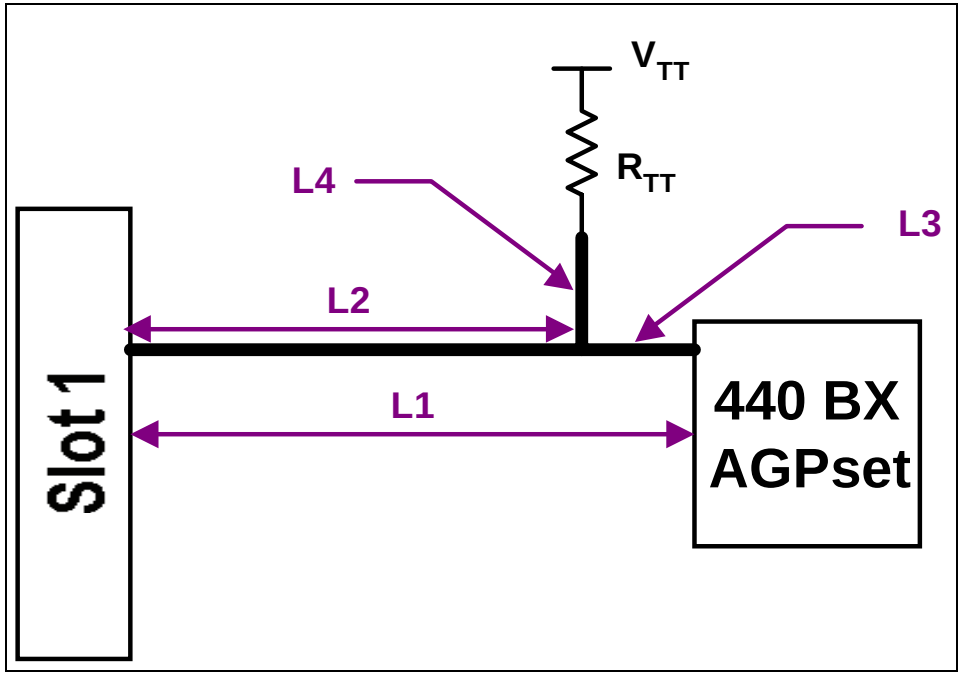


Figure 1. UP Dual Ended Termination (DET)

For UP-DET topologies (see Figure 1), the trace length L1 must be between 1.5" to 4.5" while using a 56Ω termination resistor.

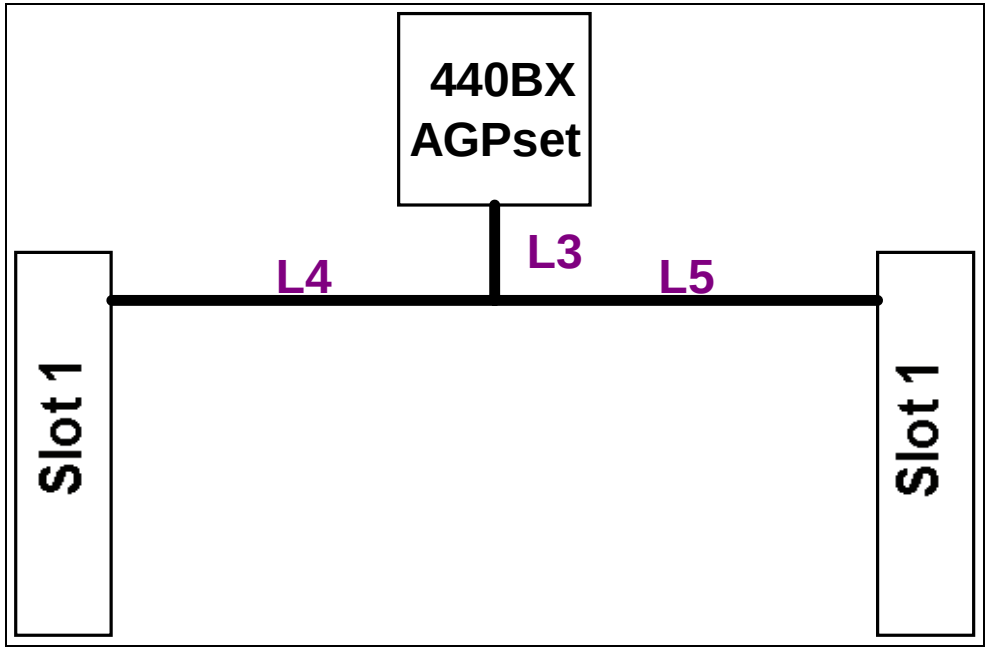


Figure 2. DP Dual Ended Termination (DET)

For DP topologies (see Figure 2) installed with a processor and a termination card, L4 + L3 or L5 + L3 must be less than or equal to 4.5". Additionally, for DP platforms with one processor installed, the termination card should be placed in the longer leg.
STATUS: For the steppings affected see the Summary Table of Changes at the beginning of this section.

43. FP Data Operand Pointer May Not Be Zero After Power On or Reset

PROBLEM: The FP Data Operand Pointer, as specified, should be reset to zero upon power on or Reset by the processor. Due to this erratum, the FP Data Operand Pointer may be nonzero after power on or Reset.
IMPLICATION: Software which uses the FP Data Operand Pointer and count on its value being zero after power on or Reset without first executing an FINIT/FNINIT instruction will use an incorrect value, resulting on incorrect behavior of the software.
WORKAROUND: Software should follow the recommendation in Section 8.2 of the *Intel Architecture Software Developer's Manual, Volume 3: System Programming Guide* (Order Number 243192). This recommendation states that if the FPU will be used,

software-initialization code should execute an FINIT/FNINIT instruction following a hardware reset. This will correctly clear the FP Data Operand Pointer to zero.
STATUS: For the steppings affected see the Summary Table of Changes at the beginning of this section.

44. MOVD Following Zeroing Instruction Can Cause Incorrect Result

PROBLEM: An incorrect result may be calculated after these circumstances:
1. A register has been zeroed with either a SUB reg, reg instruction or an XOR reg, reg instruction,
2. A value is moved with sign extension into the same register's lower 16 bits,
3. This register is then copied to an MMX™ register using the MOVD instruction prior to any other operations on the sign-extended value.

Specifically, the sign may be incorrectly extended into bits 16-31 of the MMX register. Only the MMX register is affected by this erratum.
The erratum only occurs when the 3 following steps occur in the order shown. The erratum may occur with up to 40 intervening instructions that do not modify the sign-extended value between steps 2 and 3.
1. XOR EAX, EAX
   or SUB EAX, EAX
2. MOVSX AX, BL
   or MOVSX AX, byte ptr <memory address>
   or MOVSX AX, BX
   or MOVSX AX, word ptr <memory address>
   or CBW
3. MOVD MM0, EAX

Note that this erratum may occur with “EAX” replaced with any 32-bit general purpose register, and “AX” with the corresponding 16-bit version of that replacement. “BL” or “BX” can be replaced with any 8-bit or 16-bit general purpose register. The CBW instruction is specific to the EAX register only.
In the example, EAX is forced to contain 0 by the XOR or SUB instructions. Since the four types of the MOVSX instructions and the CBW instruction modify only bits 15:8 of EAX by sign extending the lower 8 bits of EAX, bits 31:16 of EAX should always contain 0. This implies that when MOVD copies EAX to MM0, bits 31:16 of MM0 should also be 0. Under certain scenarios, bits 31:16 of MM0 are not 0, but are replicas of bit 15 (the 16th bit) of AX. This is noticeable when the value in AX after the MOVSX or CBW instruction is negative, i.e., bit 15 of AX is a 1.
When AX is positive (bit 15 of AX is a 0), MOVD will always produce the correct answer. If AX is negative (bit 15 of AX is a 1), MOVD may produce the right answer

or the wrong answer depending on the point in time when the MOVD instruction is executed in relation to the MOVSX or CBW instruction.

IMPLICATION: The effect of incorrect execution will vary from unnoticeable, due to the code sequence discarding the incorrect bits, to an application failure. If the MMX application in which MOVD is used to manipulate pixels, it is possible for one or more pixels to exhibit the wrong color or position momentarily. It is also possible for a computational application that uses the MOVD instruction in the manner described above to produce incorrect data. Note that this data may cause an unexpected page fault or general protection fault.

WORKAROUND: There are two possible workarounds for this erratum:

1. Rather than using the MOVSX-MOVD or CBW-MOVD pairing to handle one variable at a time, use the sign extension capabilities (PSRAW, etc.) within MMX™ for operating on multiple variables. This would result in higher performance as well.
2. Insert another operation that modifies or copies the sign-extended value between the MOVSX/CBW instruction and the MOVD instruction as in the example below:

```
XOR EAX, EAX (or SUB EAX, EAX)
MOVSX AX, BL (or other MOVSX or CBW instruction)
*MOV EAX, EAX
MOVD MM0, EAX
```

*Note: MOV EAX, EAX is used here as it is fairly unobtrusive. Again, EAX can be any 32-bit register.

STATUS: For the steppings affected see the Summary Table of Changes at the beginning of this section.

45. Premature Execution of a Load Operation Prior to Exception Handler Invocation

PROBLEM: This erratum can occur with any of the following situations:

1. If an instruction that performs a memory load causes a code segment limit violation
2. If a waiting floating-point instruction or MMX™ instruction that performs a memory load has a floating-point exception pending
3. If an MMX instruction that performs a memory load and has either CR0.EM =1 (Emulation bit), or a floating-point Top-of-Stack (FP TOS) not equal to 0, or a DNA exception pending

With any of the above circumstances, it is possible that the load portion of the instruction will have prematurely executed before the exception handler is entered.

IMPLICATION: In normal code execution where the target of the load operation is to write back memory there is no impact from the load being prematurely executed, nor from the restart and subsequent re-execution of that instruction by the exception handler. If the target of the load is to uncached memory that has a system side effect, restarting the instruction may cause unexpected system behavior.
WORKAROUND: Code that loads from memory that has side effects can effectively workaround this behavior by using simple integer based load instructions when accessing side effect memory and by ensuring that all code is written such that a code segment limit violation cannot occur as a part of reading to side effect memory.
STATUS: For the steppings affected see the Summary Table of Changes at the beginning of this section.

46. Read Portion of RMW Instruction May Execute Twice

PROBLEM: When the Pentium II processor executes a read-modify-write (RMW) arithmetic instruction, with memory as the destination, it is possible for a page fault to occur during the execution of the store on the memory operand after the read operation has completed but before the write operation completes.
If the memory targeted for the instruction is UC (uncached), memory will observe the occurrence of the initial load before the page fault handler and again if the instruction is restarted.
IMPLICATION: This erratum has no effect if the memory targeted for the RMW instruction causes no side effects, other than the effect that the memory location is actually read twice. If, however, the load targets a memory region that is associated with read side-effects, multiple occurrences of the initial load may cause unpredictable system behavior.
WORKAROUND: Hardware and software developers who write device drivers for custom hardware which may use a side-effect style of design should use simple loads and simple stores to transfer data to and from the device.
STATUS: For the steppings affected see the Summary Table of Changes at the beginning of this section.

47. Test Pin Must Be High During Power Up

PROBLEM: The Pentium II processor uses the PWRGOOD signal to ensure that no voltage sequencing issues arise; no pin assertions should cause the processor to change its behavior until this signal is asserted, when all power supplies and clocks to the processor are valid and stable. However, if the TESTHI signal is at a low voltage level

when the core power supply comes up, it will cause the processor to enter an invalid test state.

IMPLICATION: If this erratum occurs, the system may boot normally however, L2 cache may not be initialized.

WORKAROUND: Ensure that the 2.5 V($V_{CC2.5}$) power supply ramps at or before the 2.0 V(V_{CCCORE}) power plane. If 2.5 V ramps after core, pull up TESTHI to 2.5 V($V_{CC2.5}$) with a 100K ohm resistor. The internal pull-up will keep the signal from being asserted during power up. Alternatively, TESTHI could be pulled up to 2.0 V(V_{CCCORE}) using a 1K-10K ohm resistor.

STATUS: For the steppings affected see the Summary Table of Changes at the beginning of this section.

48. Intervening Writeback May Occur During Split-Lock

PROBLEM: During a transaction which has the LOCK# signal asserted and crosses a cache-line boundary (a.k.a. a split-lock transaction), there is a potential for an explicit writeback caused by a previous transaction to complete while the bus is locked. The explicit writeback will only be issued by the processor which has locked the bus, and the lock signal will not be deasserted until the locked transaction completes, but the atomicity of a split-lock may be compromised by this erratum. Note that the explicit writeback is an expected cycle, and no memory ordering violations will occur. This erratum is, however, a violation of the split-lock protocol.

IMPLICATION: A chipset or third-party agent (TPA) which tracks bus transactions in such a way that split-lock transactions may only consist of a read-read-write-write locked sequence, with no transactions intervening, may lose synchronization of state due to the intervening explicit writeback. Systems using chipsets or TPAs which can accept the intervening transaction will not be affected.

WORKAROUND: The bus tracking logic of all devices on the system bus should allow for the occurrence of an intervening transaction during a split-lock transaction.

STATUS: For the steppings affected see the Summary Table of Changes at the beginning of this section.

49. MC2_STATUS MSR Has Model-Specific Error Code and Machine Check Architecture Error Code Reversed

PROBLEM: The *Intel Architecture Software Developer's Manual, Volume 3: System Programming Guide*,documents that for the MC*i*_STATUS MSR, bits 15:0 contain the MCA (machine-check architecture) error code field and bits 31:16 contain the model-

specific error code field. However, for the MC2_STATUS MSR, these bits have been reversed. For the MC2_STATUS MSR, bits 15:0 contain the model-specific error code field and bits 31:16 contain the MCA error code field.
IMPLICATION: A machine check error may be decoded incorrectly if this erratum on the MC2_STATUS MSR is not taken into account.
WORKAROUND: When decoding the MC2_STATUS MSR, reverse the two error fields.
STATUS: For the steppings affected see the Summary Table of Changes at the beginning of this section.

50. Mixed Cacheability of Lock Variables Is Problematic in MP Systems

PROBLEM: This errata only affects multiprocessor systems where a lock variable address is marked cacheable in one processor and uncacheable in any others. The processors which have it marked uncacheable may stall indefinitely when accessing the lock variable. The stall is only encountered if:

- One processor has the lock variable cached, and is attempting to execute a cache lock.
- If the processor which has that address cached has it cached in its L2 only.
- Other processors, meanwhile, issue back to back accesses to that same address on the bus.

IMPLICATION: MP systems where all processors either use cache locks or consistent locks to uncacheable space will not encounter this problem. If, however, a lock variable's cacheability varies in different processors, and several processors are all attempting to perform the lock simultaneously, an indefinite stall may be experienced by the processors which have it marked uncacheable in locking the variable (if the conditions above are satisfied). Intel has only encountered this problem in focus testing with artificially generated external events. Intel has not currently identified any commercial software which exhibits this problem.
WORKAROUND: Follow a homogenous model for the memory type range registers (MTRRs), ensuring that all processors have the same cacheability attributes for each region of memory; do not use locks whose memory type is cacheable on one processor, and uncacheable on others. Avoid page table aliasing, which may produce a nonhomogenous memory model.
STATUS: For the steppings affected see the Summary Table of Changes at the beginning of this section.

51. Thermal Sensor May Assert SMBALERT# Incorrectly

PROBLEM: The Mobile Pentium II processor and the Pentium II processor Mobile Module have a thermal sensor that monitors the processor core's temperature. Please note that desktop systems could have a similar thermal device. The thermal sensor asserts SMBALERT# if the processor temperature exceeds the temperature limits set in the Alarm Threshold Registers (T_{HIGH}, T_{LOW}). It also sets the corresponding Status Register bits to identify the cause of the interrupt. Figure 1 gives one example of the how the SMBALERT# signal could be used in a system.

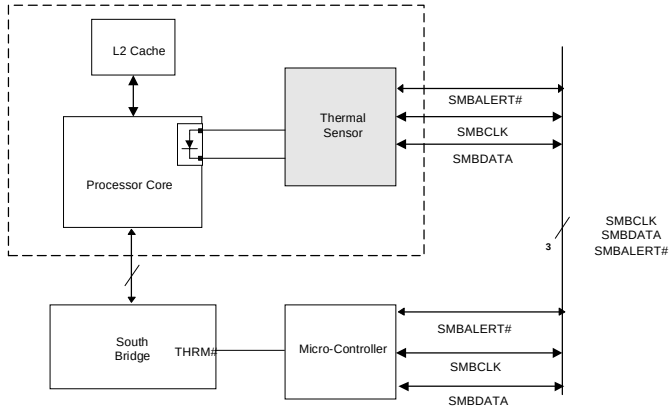


Figure 1. An Example of Micro-controller Driven Thermal Management

Under the conditions described below, the thermal sensor incorrectly generates the SMBALERT# interrupt. *All* of the following conditions must be met to trigger a false interrupt:

1. The thermal sensor must be in auto-convert mode.
2. The absolute value of the difference between the current temperature reading and the T_{HIGH} or T_{LOW} limit value must be less than or equal to 8 °C.
3. The current temperature reading must be different from the previous reading.

With a false assertion of SMBALERT#, the corresponding bit in the Status Register (L_{HIGH}, L_{LOW}, R_{HIGH}, and R_{LOW}) also will be incorrect.

IMPLICATION: There is no system impact from this erratum if temperature polling is used for processor thermal management. If the SMABLERT# interrupt is employed to manage processor thermal sensing, then servicing the false interrupt may result in premature system action depending on the software and hardware implementations

used. The rate of the false interrupts is less than the auto-convert rate of the thermal sensor.

WORKAROUND: Three different (mutually exclusive) workarounds are possible:

1. Before servicing an interrupt from the thermal sensor, read and compare the processor thermal reading with the threshold limits (T_{HIGH} or T_{LOW}). Figures 2 and 3 provide basic flowcharts for the implementation of this workaround in an interrupt driven system.
2. If the firmware implemented polls the Status Register only, then before taking any action, re-read the temperature register and do a comparison with the alarm threshold limits (T_{HIGH} or T_{LOW}) to determine if the value is actually still within the temperature window.
3. Use a temperature polling scheme to monitor the processor temperature.

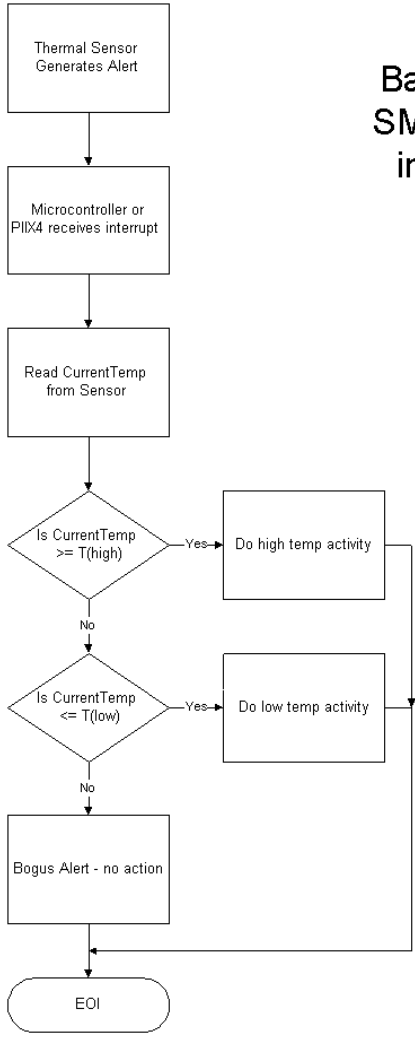


Figure 2. Workaround Flowchart: SMBALERT#-Driven System

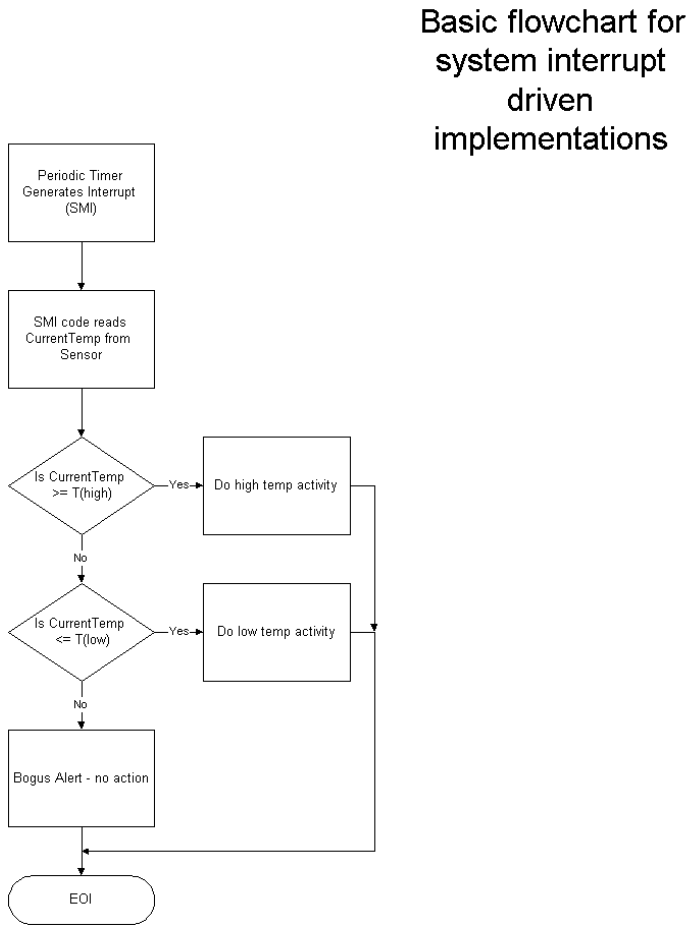


Figure 3. Workaround Flowchart: SMI#-Driven System

STATUS: For the steppings affected see the Summary Table of Changes at the beginning of this section.

1AP. APIC Access to Cacheable Memory Causes SHUTDOWN

PROBLEM: APIC operations which access memory with any type other than uncacheable (UC) are illegal. If an APIC operation to a memory type other than UC occurs and Machine Check Exceptions (MCEs) are disabled, the processor will enter SHUTDOWN after such an access. If MCEs are enabled, an MCE will occur. However, in this

circumstance, a second MCE will be signaled. The second MCE signal will cause the Pentium II processor to enter SHUTDOWN.
IMPLICATION: Recovery from a PIC access to cacheable memory will not be successful. Software that accesses only UC type memory during APIC operations will not encounter this erratum.
WORKAROUND: Ensure that the memory space to which PIC accesses can be made is marked as type UC (uncacheable) in the memory type range registers (MTRRs) to avoid this erratum.
STATUS: For the steppings affected see the Summary Table of Changes at the beginning of this section.

2AP. 2-Way MP Systems May Hang Due to Catastrophic Errors During BSP Determination

PROBLEM: In 2-way MP systems, a catastrophic error during the bootstrap processor (BSP) determination process should cause the assertion of IERR#. If the catastrophic error is due to the APIC data bus being stuck at electrical zero, then the system hangs without asserting IERR#.
IMPLICATION: 2-way MP systems may hang during boot due to a catastrophic error. This erratum has not been observed to date in a typical commercial system, but was found during focused system testing using a grounded APIC data bus.
WORKAROUND: None identified at this time.
STATUS: For the steppings affected see the Summary Table of Changes at the beginning of this section.

3AP. Write to Mask LVT (Programmed as EXTINT) Will Not Deassert Outstanding Interrupt

PROBLEM: If the APIC subsystem is configured in Virtual Wire Mode implemented through the local APIC (i.e., the 8259 INTR signal is connected to LINT0 and LVT1's interrupt delivery mode field is programmed as EXTINT), a write to LVT1 intended to mask interrupts will not deassert the internal interrupt source if the external LINT0 signal is already asserted. The interrupt will be erroneously posted to the Pentium II processor despite the attempt to mask it via the LVT.
IMPLICATION: Because of the masking attempt, interrupts may be generated when the system software expects no interrupts to be posted.
WORKAROUND: Software can issue a write to the 8259A interrupt mask register to deassert the LINT0 interrupt level, followed by a read to the controller to ensure that the

LINT0 signal has been deasserted. Once this is ensured, software may then issue the write to mask LVT entry 1.

STATUS: For the steppings affected see the Summary Table of Changes at the beginning of this section.

DOCUMENTATION CHANGES

The Documentation Changes listed in this section apply to the *Pentium® II Processor at 233 MHz, 266 MHz, 300 MHz, and 333 MHz* datasheet, the *Pentium® Pro Family Developer's Manual, Volumes 1, 2, and 3*, the *Mobile Pentium® II Processor* datasheet (Order Number 243669), the *Pentium® II Processor Mobile Module (MMC1)* datasheet (Order Number 243667), and the *Intel Architecture Software Developer's Manual, Volumes 1, 2, and 3*. All Documentation Changes will be incorporated into a future version of the appropriate Pentium II processor documentation.

1. Invalid Arithmetic Operations and Masked Responses to Them Relative to FIST/FISTP Instruction

The *Pentium® Pro Family Developer's Manual, Volume 2: Programmer's Reference Manual*, Table 7-20, and the *Intel Architecture Software Developer's Manual, Volume 1*, Table 7-20, show "Invalid Arithmetic Operations and the Masked Responses to Them." The table entry corresponding to the FIST/FISTP condition is missing, and is shown below:

Condition	Masked Response
FIST/FISTP instruction when input operand <> MAXINT for destination operand size.	Return MAXNEG to destination operand.

When FIST/FISTP instruction is executed with input operand <> and the destination operand size is MAXINT, the floating-point zero-divide exception will return MAXNEG to the destination operand as its masked response.

2. FIDIV/FIDIVR m16int Description

The *Pentium® Pro Family Developer's Manual, Volume 2: Programmer's Reference Manual*, pages 11-118 and 11-121, and the *Intel Architecture Software Developer's Manual, Volume 1*, pages 3-118 and 3-122, show in the Description column for the FIDIV *m16int* instruction as "Divide ST(0) by *m64int* by ST(0) and store the result in ST(0)" and FIDIVR *m16int* instruction as "Divide *m64int* by ST(0) and store the result in ST(0)" In both of these cases, *m64int* should be replaced with *m16int*.

3. PUSH Does Not Pad With Zeros

The *Pentium® Pro Family Developer's Manual, Volume 2: Programmer's Reference Manual*, page 4-2, and the *Intel Architecture Software Developer's Manual, Volume 1*, page 4-3, contain a section regarding stack alignment. The last sentence in the first paragraph of this section, which reads "If a 16-bit value is pushed onto a 32-bit wide stack, the value is automatically padded with zeros out to 32-bits." should be removed. The PUSH instruction does not pad with zeros.

4. DR7, Bit 10 is Reserved

The *Pentium® Pro Family Developer's Manual, Volume 3: Operating Systems Writer's Manual*, shows Figure 10-1, "Debug Registers." Bit 10 of DR7 should be "Reserved" instead of "1."

5. Additional States That Are Not Automatically Saved and Restored

In Section 9.4.1 of the *Pentium® Pro Family Developer's Manual, Volume 3: Operating Systems Writer's Manual*, and in Section 11.4.1. of the *Intel Architecture Software Developer's Manual, Volume 3: System Programming Guide*, the end of section lists the registers that are not automatically saved and restored following an SMI and the RSM instruction, respectively. The last two paragraphs should be as follows:

The following state is not automatically saved and restored following an SMI and the RSM instruction, respectively:

- Debug registers DR0 through DR3.
- The FPU registers.
- The MTRRs.
- Control register CR2.
- The model-specific registers (for the P6 family and Pentium® processors) or test registers TR3 through TR7 (for the Pentium and Intel486™ processors).
- The state of the trap controller.
- The machine-check architecture registers.
- The APIC internal interrupt state (ISR, IRR, etc.).
- The Microcode Update state.

If an SMI is used to power down the processor, a power-on reset will be required before returning to SMM, which will reset much of this state back to its default values. So an SMI handler that is going to trigger power down should first read these registers listed above directly, and save them (along with the rest of RAM) to nonvolatile storage.

After the power-on reset, the continuation of the SMI handler should restore these values, along with the rest of the system's state. Anytime the SMI handler changes these registers in the processor it must also save and restore them.

NOTE

A small subset of the MSRs (such as the time-stamp counter and performance-monitoring counter) are not arbitrarily writeable and therefore cannot be saved and restored. SMM-based power-down and restoration should only be performed with operating systems that do not use or rely on the values of these registers. Operating system developers should be aware of this fact and ensure that their operating-system assisted power-down and restoration software is immune to unexpected changes in these register values.

6. S.E.C. Cartridge Mechanical Specification Corrections

In Section 4.3 of *Pentium® II Processor at 233 MHz, 266 MHz, 300 MHz, and 333 MHz* datasheet, the following corrections should be made:

4.3. Thermal Solution Attach Methods

The design of the thermal plate is intended to support two different attach methods — heatsink clips and Rivscrews*. Figure 41 shows the thermal plate and the locations of the attach features. Only one attach method should be used for any thermal solution.

4.3.1 HEATSINK CLIP ATTACH

Figure 23 and Figure 24 illustrate example clip designs to support a low profile and a full height heatsink, respectively. The clips attach the heatsink by engaging with the underside of the thermal plate. The clearance of the thermal plate to the internal processor substrate is a minimum **0.089**" (illustrated in Figure 23 and Figure 24). The clips should be designed such that they will engage within this space, and also not damage the substrate upon insertion or removal. Finally, the clips should be able to retain the heatsink onto the thermal plate through a system level mechanical shock and vibration test. The clips should also apply a high enough force to spread the interface material for the spot size selected.

4.3.2. RIVSCREW* ATTACH

The Rivscrew attach mechanism uses a specialized rivet that is inserted through a hole in the heatsink into the thermal plate. Upon insertion, a threaded fastener is formed that can be removed if necessary. For Rivscrew attachment, the minimum gap between the thermal plate and the processor substrate is **0.125**". For use of the Advel Rivscrew (part number 1712-3510), the heatsink base thickness must be 0.140 ±0.010". See Figure 25, Figure 26 and Figure 27 for details of heatsink requirements for use with Rivscrews.

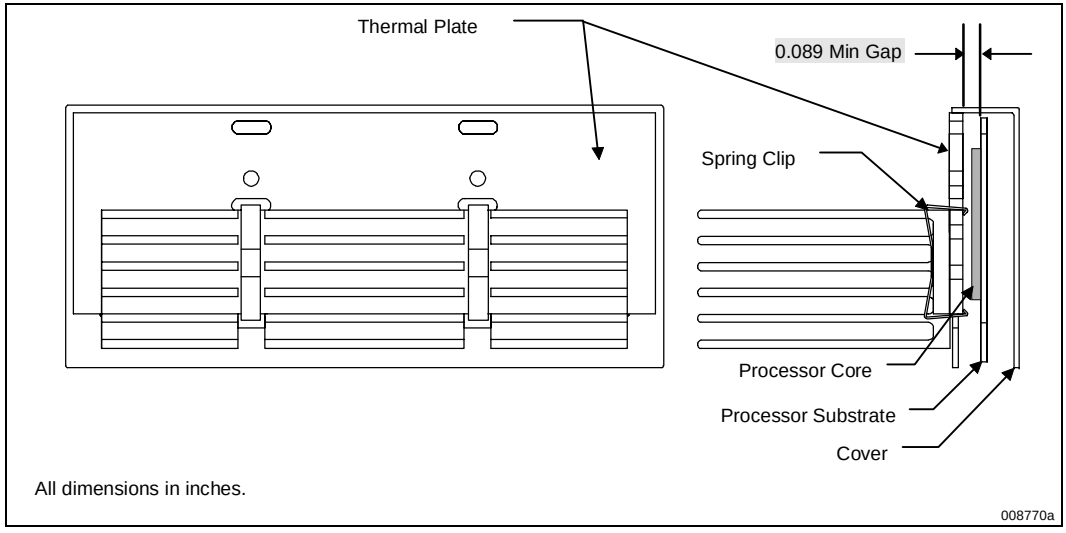


Figure 23. Processor with an Example Low Profile Heatsink Attached using Spring Clips

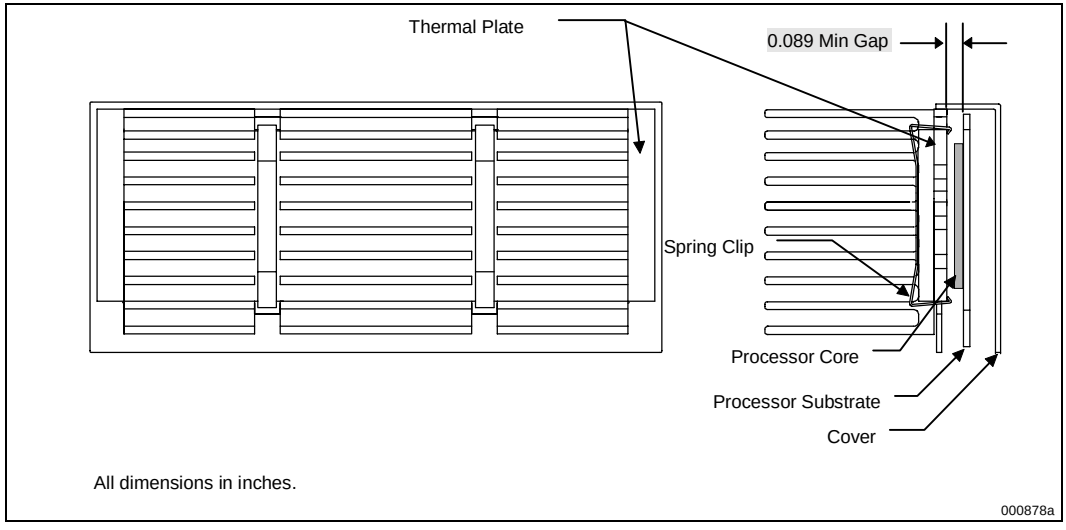


Figure 24. Processor with an Example Full Height Heatsink Attached using Spring Clips

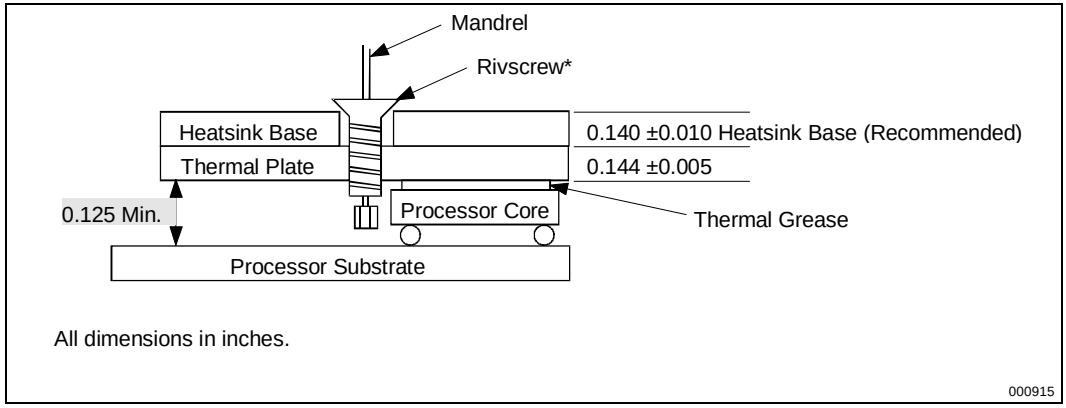


Figure 26. Hfeatsink Rivscrew* and Thermal Plate Recommendations and Guidelines

7. Cache and TLB Description Correction

In the *Pentium® Pro Family Developer's Manual, Volume 2: Programmer's Reference Manual*, Table 11-10, and in the *Intel Architecture Software Developer's Manual, Volume 2: Instruction Set Reference*, Table 3-7, the correct description for descriptor value 02H should be as follows:

Descriptor Value	Cache or TLB Description
02H	Instruction TLB: 4M-Byte Pages, **fully** associative, **2** entries

Also, the third bullet after the table should be as follows:

I. Bytes 1, 2, and 3 of register EAX indicate that the processor contains the following:
   - 01H–A 32-entry instruction TLB (4-way set associative) for mapping 4-Kbytes pages.
   - 02H–A **2**-entry instruction TLB (**fully** associative) for mapping 4-Mbyte pages.
   - 03H–A 64-entry data TLB (4-way set associative) for mapping 4-Kbyte pages.

Finally, for the *Pentium® Pro Family Developer's Manual, Volume 3: Operating Systems Writer's Manual,* Table 11-1, the following corrections should be made:

Cache or Buffer	Characteristics
Instruction TLB (Large Pages)	**2** entries, **fully** associative

For the *Intel Architecture Software Developer's Manual, Volume 3: System Programming Guide,* Table 9-1, the following corrections should be made:

Cache or Buffer	Characteristics
Instruction TLB (Large Pages)	- P6 family processors: **2** entries, **fully** associative - Pentium® processor: Uses same TLB as used for 4-Kbyte pages - Intel486™ processor: None (large pages not supported)
Data TLB (Large Pages)	- P6 family processors: 8 entries, 4-way set associative - Pentium processor: **8** entries, **4**-way set associative; uses same TLB as used for 4-Kbyte pages in Pentium processors with MMX™ technology - Intel486 processor: None (large pages not supported)

8. SMRAM State Save Map Contains Documentation Errors

In the *Pentium® Pro Family Developer's Manual, Volume 3, Operating System Writer's Manual*, Chapter 9, "System Management Mode," Table 9-1 incorrectly documents the SMBASE+Offset for IDT Base and GDT Base on the Pentium II processor. In the *Intel Architecture Software Developer's Manual, Volume 3, System Programming Guide*, Chapter 11, "System Management Mode," Table 11-1 incorrectly documents the SMBASE+Offset for IDT Base and GDT Base for Pentium II processors.
The storage locations for these parameters are model specific (i.e., they may differ between the Pentium processor, the Pentium Pro processor, Pentium II processor, and other P6 family proliferations). These entries in the tables above will be changed to Reserved. Hardware and software may not rely on the contents of these Reserved regions.

SPECIFICATION CLARIFICATIONS

The Specification Clarifications listed in this section apply to the *Pentium® II Processor at 233 MHz, 266 MHz, 300 MHz and 333 MHz* datasheet, the *Pentium® Pro Family Developer's Manual, Volumes 1, 2, and 3*, the *Mobile Pentium® II Processor* datasheet (Order Number 243669), the *Pentium® II Processor Mobile Module (MMC1)* datasheet (Order Number 243667), and the *Intel Architecture Software Developer's Manual, Volumes 1, 2, and 3*. All Specification Clarifications will be incorporated into a future version of the appropriate Pentium II processor documentation.

1. Writes to WC Memory

Section 11.3 of the *Pentium® Pro Family Developer's Manual, Volume 3: Operating System Writer's Guide*, identifies that "Writes" to a region of WC memory "may be delayed and combined in the write buffer to reduce memory accesses." This sentence should state that "Writes" to a region of WC memory "may be delayed and combined in the write buffer to reduce memory accesses. The writes may be delayed until the next occurrence of a buffer or processor serialization event, e.g., CPUID execution, a read or write to uncached memory, interrupt occurrence, LOCKed instruction execution, etc., if the WC buffer is partially filled."

2. Multiple Processors Protocol and Restrictions

Section 7.5.2, "Protocol Requirements and Restrictions," in the *Pentium® Pro Family Developer's Manual, Volume 3: Operating System Writer's Guide*, contain inconsistencies which will be clarified as follows:

7.5.2 Protocol Requirements and Restrictions

The MP protocol imposes the following requirements and restrictions on the system:

- An APIC clock (APICLK) must be provided on all systems based on the Pentium® Pro processor.
- All interrupt mechanisms must be disabled for the duration of the MP protocol algorithm including the window of time between the assertion of INIT# or receipt of an INIT IPI by the application processors and the receipt of a STARTUP IPI by the application processors. That is, requests generated by interrupting devices must not be seen by the local APIC unit (on board the processor) until the completion of the algorithm. Failure to disable the interrupt mechanisms may result in processor shutdown.

- The MP protocol should be initiated only after a hardware reset. After completion of the protocol algorithm, a flag is set in the APIC base MSR of the BSP (APIC_BASE.BSP) to indicate that it is the BSP. This flag is cleared for all other processors. If a processor or the system is subject to an INIT sequence (either through the INIT# pin or an INIT IPI), then the MP protocol is not reexecuted. Instead, each processor examines its BSP flag to determine whether the processor should boot or wait for a STARTUP IPI.

3. NMI Handling While in SMM

Section 9.7, "NMI Handling While in SMM," in the *Pentium® Pro Family Developer's Manual, Volume 3: Operating System Writer's Guide*, will be clarified as follows:

9.7 NMI Handling While in SMM

NMI interrupts are blocked upon entry to the SMI handler. If an NMI request occurs during the SMI handler, it is latched and serviced after the processor exits SMM. Only one NMI request will be latched during the SMI handler. If an NMI request is pending when the processor executes the RSM instruction, the NMI is serviced before the next instruction of the interrupted code sequence.

Although the NMI requests are blocked when the CPU enters SMM, they may be enabled through software by executing an IRET/IRETD instruction. If the SMM handler requires the use of NMI interrupts, it should invoke a dummy interrupt service routine for the purpose of executing an IRET/IRETD instruction. Once an IRET/IRETD instruction is executed, NMI interrupt requests are serviced in the same "real mode" manner in which they are handled outside of SMM.

4. Critical Sequence of Events During a Page Fault Exception

Section 3.6.4, "Page-Directory and Page-Table Entries," in the *Pentium® Pro Family Developer's Manual, Volume 3: Operating System Writer's Guide*, will be clarified as follows:

If the processor generates a page-fault exception, the operating system must carry out the following operations in this order:

1. Copy the page from disk storage into physical memory if needed.
2. Load the page address into the page-table or page-directory entry and set its **present** flag. Other bits, such as the dirty and accessed bits, may also be set at this time.

3. Invalidate the current page table entry in the TLB (see Section 3.7, “Translation Lookaside Buffers (TLBs)” for a discussion of TLBs and how to invalidate them).
4. Return from the page fault handler to restart the interrupted program or task.

5. Performance-Monitoring Counter Issues

The following table documents the characterized differences between the behavior of the Pentium II processor’s performance-monitoring counters and that documented in Appendix B, “Performance Monitoring Counters,” of the *Pentium® Pro Family Developer’s Manual, Volume 3: Operating System Writer’s Guide.*
The following table replaces Table B-1, Appendix B, *Pentium® Pro Family Developer’s Manual, Volume 3.* The only change to this new table are enhanced descriptions of the events counted.

Unit	Event Number	Mnemonic Event Name	Unit Mask	Description	Comments
Data Cache Unit (DCU)	43H	DATA_ MEM_ REFS	00H	All loads from any memory type. All stores to any memory type. Each part of a split is counted separately. The internal logic counts not only external memory loads and stores, but also internal retries. Note: 80 bit floating-point accesses are double counted, since they are decomposed into a 16 bit exponent load and a 64 bit mantissa load. Memory accesses are only counted when they are actually performed. E.g., a load that gets squashed because a	

Unit	Event Number	Mnemonic Event Name	Unit Mask	Description	Comments
				previous cache miss is outstanding to the same address, and which finally gets performed, is only counted once. Does not include I/O accesses, or other nonmemory accesses.	
	45H	DCU_LINES_IN	00H	Total lines allocated in the DCU.	
	46H	DCU_M_LINES_IN	00H	Number of M state lines allocated in the DCU.	
	47H	DCU_M_LINES_OUT	00H	Number of M state lines evicted from the DCU. This includes evictions via snoop HITM, intervention or replacement.	
	48H	DCU_MISS_OUTSTANDING	00H	Weighted number of cycles while a DCU miss is outstanding, incremented by the number of outstanding cache misses at any particular time. Cacheable read requests only are considered. Uncacheable requests are excluded. Read-for-ownerships are counted as well as line fills, invalidates, and stores.	An access that also misses the L2 is short-changed by 2 cycles. (i.e., if count is N cycles, should be N+2 cycles.) Subsequent loads to the same cache line will not result in any additional counts. Count value not precise, but still useful.

Unit	Event Number	Mnemonic Event Name	Unit Mask	Description	Comments
Instr-uction Fetch Unit (IFU)	80H	IFU_ IFETCH	00H	Number of instruction fetches, both cacheable and noncacheable. Including UC fetches.	
	81H	IFU_ IFETCH_MISS	00H	Number of instruction fetch misses. All instruction fetches that do not hit the IFU, i.e., that produce memory requests. Includes UC accesses.	
	85H	ITLB_ MISS	00H	Number of ITLB misses.	
	86H	IFU_ MEM_ STALL	00H	Number of cycles instruction fetch is stalled, for any reason. Includes IFU cache misses, ITLB misses, ITLB faults and other minor stalls.	
	87H	ILD_ STALL	00H	Number of cycles that the instruction length decoder is stalled.	
L2 Cache[1]	28H	L2_ IFETCH	MESI 0FH	Number of L2 instruction fetches. This event indicates that a normal instruction fetch was received by the L2. The count includes only L2 cacheable instruction fetches; it does not include	

Unit	Event Number	Mnemonic Event Name	Unit Mask	Description	Comments
				UC instruction fetches. It does not include ITLB miss accesses.	
	29H	L2_LD	MESI 0FH	Number of L2 data loads. This event indicates that a normal, unlocked, load memory access was received by the L2. It includes only L2 cacheable memory accesses; it does not include I/O accesses, other nonmemory accesses, or memory accesses such as UC/WT memory accesses. It does include L2 cacheable TLB miss memory accesses.	
	2AH	L2_ST	MESI 0FH	Number of L2 data stores. This event indicates that a normal, unlocked, store memory access was received by the L2. Specifically, it indicates that the DCU sent a read-for-ownership request to the L2. It also includes Invalid to Modified requests sent by the DCU to the L2. It includes only L2 cacheable store memory accesses; it	

Unit	Event Number	Mnemonic Event Name	Unit Mask	Description	Comments
				does not include I/O accesses, other nonmemory accesses, or memory accesses like UC/WT stores. It includes TLB miss memory accesses.	
	24H	L2_LINES _IN	00H	Number of lines allocated in the L2.	
	26H	L2_LINES_ OUT	00H	Number of lines removed from the L2 for any reason.	
	25H	L2_M_ LINES_ INM	00H	Number of modified lines allocated in the L2.	
	27H	L2_M_ LINES_ OUTM	00H	Number of modified lines removed from the L2 for any reason.	
	2EH	L2_ RQSTS	MESI 0FH	Total number of L2 requests.	
	21H	L2_ADS	00H	Number of L2 address strobes.	
	22H	L2_DBUS_ BUSY	00H	Number of cycles during which the L2 cache data bus was busy.	
	23H	L2_DBUS_ BUSY_ RD	00H	Number of cycles during which the data bus was busy transferring read data from L2 to the processor.	
External Bus Logic	62H	BUS_ DRDY_ CLOCKS	00H (Self) 20H	Number of clocks during which DRDY# is asserted. Essentially,	Unit Mask = 00H counts bus clocks when the processor is driving DRDY#.

Unit	Event Number	Mnemonic Event Name	Unit Mask	Description	Comments
(EBL)[2]			(Any)	utilization of the external system data bus.	Unit Mask = 20H counts in processor clocks when any agent is driving DRDY#.
	63H	BUS_ LOCK_ CLOCKS	00H (Self) 20H (Any)	Number of clocks during which LOCK# is asserted on the external system bus.	Always counts in processor clocks.
	60H	BUS_REQ_ OUT- STAND- ING	00H (Self)	Number of bus requests outstanding. This counter is incremented by the number of cacheable read bus requests outstanding in any given cycle.	Counts only DCU full-line cacheable reads, not RFOs, writes, instruction fetches, or anything else. Counts "waiting for bus to complete" (last data chunk received).
	65H	BUS_ TRAN_ BRD	00H (Self) 20H (Any)	Number of burst read transactions.	
	66H	BUS_ TRAN_ RFO	00H (Self) 20H (Any)	Number of completed read for ownership transactions.	
	67H	BUS_ TRANS_W B	00H (Self) 20H (Any)	Number of completed write back transactions.	
	68H	BUS_ TRAN_ IFETCH	00H (Self) 20H (Any)	Number of completed instruction fetch transactions.	

Unit	Event Number	Mnemonic Event Name	Unit Mask	Description	Comments
	69H	BUS_ TRAN_ INVAL	00H (Self) 20H (Any)	Number of completed invalidate transactions.	
	6AH	BUS_ TRAN_ PWR	00H (Self) 20H (Any)	Number of completed partial write transactions.	
	6BH	BUS_ TRANS_P	00H (Self) 20H (Any)	Number of completed partial transactions.	
	6CH	BUS_ TRANS_ IO	00H (Self) 20H (Any)	Number of completed I/O transactions.	
	6DH	BUS_ TRAN_ DEF	00H (Self) 20H (Any)	Number of completed deferred transactions.	
	6EH	BUS_ TRAN_ BURST	00H (Self) 20H (Any)	Number of completed burst transactions.	
	70H	BUS_ TRAN_ ANY	00H (Self) 20H (Any)	Number of all completed bus transactions. Address bus utilization can be calculated knowing the minimum address bus occupancy. Includes special cycles , etc.	
	6FH	BUS_ TRAN_ MEM	00H (Self) 20H (Any)	Number of completed memory transactions.	

Unit	Event Number	Mnemonic Event Name	Unit Mask	Description	Comments
	64H	BUS_ DATA_ RCV	00H (Self)	Number of bus clock cycles during which this processor is receiving data.	
	61H	BUS_BNR_ DRV	00H (Self)	Number of bus clock cycles during which this processor is driving the BNR# pin.	
	7AH	BUS_HIT_ DRV	00H (Self)	Number of bus clock cycles during which this processor is driving the HIT# pin.	Includes cycles due to snoop stalls. The event counts correctly, but the BPM*i* pins function as follows based on the setting of the PC bits (bit 19 in the PerfEvtSel0 and PerfEvtSel1 registers). If the core clock to bus clock ratio is 2:1 or 3:1, and a PC bit is set, the BPM*i* pins will be asserted for a single clock when the counters overflow. If the PC bit is clear, the processor toggles the BPM*i* pins when the counter overflows. If the clock ratio is not 2:1 or 3:1, the BPM*i* pins will not function for these performance-monitoring counter events.

Unit	Event Number	Mnemonic Event Name	Unit Mask	Description	Comments
	7BH	BUS_ HITM_ DRV	00H (Self)	Number of bus clock cycles during which this processor is driving the HITM# pin.	Includes cycles due to snoop stalls. The event counts correctly, but the BPM*i* pins function as follows based on the setting of the PC bits (bit 19 in the PerfEvtSel0 and PerfEvtSel1 registers). If the core clock to bus clock ratio is 2:1 or 3:1, and a PC bit is set, the BPM*i* pins will be asserted for a single clock when the counters overflow. If the PC bit is clear, the processor toggles the BPM*i* pins when the counter overflows. If the clock ratio is not 2:1 or 3:1, the BPM*i* pins will not function for these performance-monitoring counter events.
	7EH	BUS_ SNOOP_ STALL	00H (Self)	Number of clock cycles during which the bus is snoop stalled.	
Float-ing Point Unit	C1H	FLOPS	00H	Number of computational floating-point operations retired. Excludes floating-point computational	Counter 0 only

Unit	Event Number	Mnemonic Event Name	Unit Mask	Description	Comments
				operations that cause traps or assists. Includes floating-point computational operations executed by the assist handler. Includes internal sub-operations of complex floating-point instructions like transcendentals. Excludes floating-point loads and stores.	
	10H	FP_COMP_ OPS_ EXE	00H	Number of computational floating-point operations executed. The number of FADD, FSUB, FCOM, FMULs, integer MULs and IMULs, FDIVs, FPREMs, FSQRTS, integer DIVs and IDIVs. Note not the number of cycles but, the number of operations. This event does not distinguish an FADD used in the middle of a transcendental flow from a separate FADD instruction.	Counter 0 only

Unit	Event Number	Mnemonic Event Name	Unit Mask	Description	Comments
	11H	FP_ ASSIST	00H	Number of floating-point exception cases handled by microcode.	Counter 1 only. This event includes counts due to speculative execution.
	12H	MUL	00H	Number of multiplies. Note: includes integer and well FP multiplies and is speculative.	Counter 1 only
	13H	DIV	00H	Number of divides. Note: includes integer and FP multiplies and is speculative.	Counter 1 only
	14H	CYCLES_DIV_ BUSY	00H	Number of cycles that the divider is busy, and cannot accept new divides. Note: includes integer and FP divides, FPREM, FPSQRT, etc., and is speculative.	Counter 0 only
Memory Ordering	03H	LD_ BLOCKS	00H	Number of store buffer blocks. Includes counts caused by preceding stores whose addresses are unknown, preceding stores whose addresses are known to conflict, but whose data is unknown and preceding stores that conflicts with the load, but which incompletely	

Unit	Event Number	Mnemonic Event Name	Unit Mask	Description	Comments
				overlap the load.	
	04H	SB_ DRAINS	00H	Number of store buffer drain cycles. Incremented during every cycle the store buffer is draining. Draining is caused by serializing operations like CPUID, synchronizing operations like XCHG, Interrupt acknowledgment as well as other conditions such as cache flushing.	
	05H	MIS-ALIGN_ MEM_ REF	00H	Number of misaligned data memory references. Incremented by 1 every cycle during which either the processor load or store pipeline dispatches a misaligned uop. Counting is performed if its the first half or second half, or if it is blocked, squashed or misses. Note in this context misaligned means crossing a 64 bit boundary.	It should be noted that MISALIGN_MEM _REF is only an approximation, to the true number of misaligned memory references. The value returned is roughly proportional to the number of misaligned memory accesses, i.e., the size of the problem.
In-struc-tion	C0H	INST_ RETIRED	OOH	Number of instructions retired.	A hardware interrupt received during/after the last iteration of

Unit	Event Number	Mnemonic Event Name	Unit Mask	Description	Comments
De-coding and Retire-ment					the REP STOS flow causes the counter to undercount by 1 instruction.
	C2H	UOPS_ RETIRED	00H	Number of UOPs retired.	
	D0H	INST_ DECOD-ER	00H	Number of instructions decoded.	
Inter-rupts	C8H	HW_INT_ RX	00H	Number of hardware interrupts received.	
	C6H	CYCLES_ INT_ MASKED	00H	Number of processor cycles for which interrupts are disabled.	
	C7H	CYCLES_ INT_ PENDING_ AND_ MASKED	00H	Number of processor cycles for which interrupts are disabled and interrupts are pending.	
Bran-ches	C4H	BR_INST_R ETIRED	00H	Number of branch instructions retired.	
	C5H	BR_MISS_ PRED_ RETIRED	00H	Number of mispredicted branches retired.	
	C9H	BR_ TAKEN_ RETIRED	00H	Number of taken branches retired.	
	CAH	BR_MISS_P RED_ TAKEN_ RET	00H	Number of taken mispredictions branches retired.	
	E0H	BR_INST_ DECOD-ED	00H	Number of branch instructions decoded.	

Unit	Event Number	Mnemonic Event Name	Unit Mask	Description	Comments
	E2H	BTB_ MISSES	00H	Number of branches that for which the BTB did not produce a prediction	
	E4H	BR_ BOGUS	00H	Number of bogus branches.	
	E6H	BA-CLEARS	00H	Number of time BACLEAR is asserted. This is the number of times that a static branch prediction was made, where the branch decoder decided to make a branch prediction because the BTB did not.	
Stalls	A2H	RE-SOURCE_S TALLS	00H	Incremented by one during every cycle that there is a resource related stall. Includes register renaming buffer entries, memory buffer entries. Does not include stalls due to bus queue full, too many cache misses, etc., In addition to resource related stalls, this event counts some other events. Includes stalls arising during branch misprediction recovery, e.g., if	

Unit	Event Number	Mnemonic Event Name	Unit Mask	Description	Comments
				retirement of the mispredicted branch is delayed and stalls arising while store buffer is draining from synchronizing operations.	
	D2H	PARTIAL_ RAT_ STALLS	00H	Number of cycles or events for partial stalls. Note Includes flag partial stalls.	
Segment Register Loads	06H	SEGMENT_ REG_ LOADS	00H	Number of segment register loads	
Clocks	79H	CPU_CLK_ UNHALTED	00H	Number of cycles during which the processor is not halted.	

NOTES:

1. Several L2 cache events, where noted, can be further qualified using the Unit Mask (UMSK) field in the PerfEvtSel0 and PerfEvtSel1 registers. The lower 4 bits of the Unit Mask field are used in conjunction with L2 events to indicate the cache state or cache states involved. The Pentium® II processor identifies cache states using the "MESI" protocol and consequently each bit in the Unit Mask field represents one of the four states: UMSK[3] = M (8H) state, UMSK[2] = E (4H) state, UMSK[1] = S (2H) state, and UMSK[0] = I (1H) state. UMSK[3:0] = MESI (FH) should be used to collect data for all states; UMSK = 0H, for the applicable events, will result in nothing being counted.
2. All of the external bus logic (EBL) events, except where noted, can be further qualified using the Unit Mask (UMSK) field in the PerfEvtSel0 and PerfEvtSel1 registers. Bit 5 of the UMSK field is used in conjunction with the EBL events to indicate whether the processor should count transactions that are self generated (UMSK[5] = 0) or transactions that result from any processor on the bus (UMSK[5] = 1).

6. POP[ESP] with 16-bit Stack Size

In the *Pentium® Pro Family Developer's Manual, Volume 2: Programmer's Reference Manual*, and the *Intel Architecture Software Developer's Manual, Volume 2: Instruction Set Reference*, the section regarding "POP–Pop a Value from the Stack," the following note:

"If the ESP register is used as a base register for addressing a destination operand in memory, the POP instruction computes the effective address of the operand after it increments the ESP register."

is incomplete, and should read as follows:

"If the ESP register is used as a base register for addressing a destination operand in memory, the POP instruction computes the effective address of the operand after it increments the ESP register. For the case of a 16-bit stack where ESP wraps to 0h as a result of the POP instruction, the resulting location of the memory write is processor family specific."

In Section 15.12.1 of the *Pentium® Pro Family Developer's Manual, Volume 3: Operating System Writer's Guide*, and Section 17.23.1 of the *Intel Architecture Software Developer's Manual, Volume 3: System Programming Guide*, add a new section:

A POP-to-memory instruction, which uses the stack pointer (ESP) as a base register.

For a POP-to-memory instruction that meets the following conditions:

1. The stack segment size is 16-bit
2. Any 32-bit addressing form with the SIB byte specifying ESP as the base register
3. The initial stack pointer is FFFCh(32-bit operand) or FFFEh (16-bit operand) and will wrap around to 0h as a result of the POP operation

The result of the memory write is processor family specific. For example, in Pentium II and Pentium Pro processors the result of the memory write is to SS:0h plus any scaled index and displacement. In Pentium and i486™ processors, the result of the memory write may be either a stack fault (real mode or protected mode with stack segment size of 64 Kbyte), or write to SS:10000h plus any scaled index and displacement (protected mode and stack segment size exceeds 64 Kbyte).

SPECIFICATION CHANGES

The Specification Changes listed in this section apply to the *Pentium® II Processor at 233 MHz, 266 MHz, 300 MHz, and 333 MHz* datasheet, or the *Mobile Pentium® II Processor* datasheet (Order Number 243669), or the *Pentium® II Processor Mobile Module (MMC1)* datasheet (Order Number 243667). All Specification Changes will be incorporated into a future version of the appropriate Pentium II processor documentation.

The latest datasheet was published January 1998.

1. Mixing Steppings in MP Systems

This note explains issues related to populating the second slot in a Slot 1 DP platform. The Pentium II processor, model 5 has a different L2 cache initialization algorithm than Pentium II processor, model 3. Therefore it is important for BIOS to incorporate necessary Pentium II processor, model 5 updates when populating the second slot 1 with Pentium II processor, model 5 in a Pentium II processor, model 3 DP platform.

In the following table, the top row indicates the existing processor in a DP system and left-most column displays the upgrade options for the second slot. A number indicates a known issue, and refers to the numbered list under notes. An “NI” implies that there are currently no known issues associated with mixing these processors. Provided appropriate Microcode Update is loaded into the respective processors, both cases (“NI” and number(s), as shown) are supported by Intel.

Intel does not recommend running processors at unlike frequencies. An “X” implies invalid combinations based upon different frequency parts. While there are no currently known issues associated with mixing cache sizes, Intel does not recommend or validate mixing cache sizes.

Table 1: The top row indicates the existing processor in a DP system and left-most column displays the upgrade options for the second slot

	SL264/ 233 MHz	SL265/ 266 MHz	SL268/ 233 MHz	SL269/ 266 MHz	SL28R/ 300 MHz	SL2HD/ 233 MHz	SL2HC/ 266 MHz	SL2HF/ 233 MHz	SL2HE/ 266 MHz	SL2HA/ 300 MHz	SL2KA/ 333 MHz
SL264/ 233 MHz	2	X	1, 2	X	X	2	X	1, 2	X	X	X
SL265/ 266 MHz	X	2	X	1, 2	X	X	3	X	1, 2	X	X
SL268/ 233 MHz	1, 2	X	2	X	X	1, 2	X	2	X	X	X
SL269/ 266 MHz	X	1, 2	X	2	X	X	1, 2	X	2	X	X
SL28R/ 300 MHz	X	X	X	X	2	X	X	X	X	2	X
SL2HD/ 233 MHz	2	X	1, 2	X	X	NI	X	1	X	X	X
SL2HC/ 266 MHz	X	2	X	1, 2	X	X	NI	X	1	X	X
SL2HF/ 233 MHz	1, 2	X	2	X	X	1	X	NI	X	X	X
SL2HE/ 266 MHz	X	1, 2	X	2	X	X	1	X	NI	X	X
SL2HA/ 300 MHz	X	X	X	X	2	X	X	X	X	NI	X
SL2KA/ 333 MHz	X	X	X	X	X	X	X	X	X	X	3

NOTES:

1. Intel DOES NOT recommend mixing ECC and non-ECC processors. However, if mixed, the system BIOS should disable the ECC on the ECC supported processor to ensure consistent behavior of time-dependent software, e.g., timing loops.
2. Errata 16 and 17 may be problematic for DP systems which use Pentium® II processor, model 3 C0 stepping.
3. All processors must have their appropriate microcode updates loaded.

2. System Bus Timings Changes

In Table 12 and Table 13 of *Pentium® II Processor at 233 MHz, 266 MHz, 300 MHz, and 333 MHz* datasheet, the following changes should be made:

Table 12. GTL+ Signal Groups System Bus AC Specifications[1, 2]

T#	Parameter	Min	Max	Unit	Figure	Notes
T7:	GTL+ Output Valid Delay	1.07	6.37	ns	8	3
T8:	GTL+ Input Setup Time	**1.96**		ns	9	4, 5, 6
T9:	GTL+ Input Hold Time	1.53		ns	9	7
T10:	RESET# Pulse Width	1.00		ms	12	8

NOTES:

1. Not 100% tested. Specified by design characterization.
2. All AC timings for the GTL+ signals are referenced to the BCLK rising edge at 0.70 V at the processor edge fingers. All GTL+ signal timings (address bus, data bus, etc.) are referenced at 1.00 V at the processor edge fingers.

3. Valid delay timings for these signals are specified into 50 W to 1.5 V.
4. A minimum of 3 clocks must be specified between two active-to-inactive transitions of TRDY#.
5. RESET# can be asserted (active) asynchronously, but must be deasserted synchronously.
6. Specification is for a minimum 0.40 V swing.
7. Specification is for a maximum 1.0 V swing.
8. After Vcc_{CORE}, Vcc_{L2} and BCLK become stable.

Table 13. System Bus AC Specifications (CMOS Signal Group)[1, 2, 3]

T# Parameter	Min	Max	Unit	Figure	Notes
T11: 2.5 V Output Valid Delay	1.00	10.5	ns	8	4
T12: 2.5 V Input Setup Time	**4.50**		ns	9	5, 6
T13: 2.5 V Input Hold Time	**1.50**		ns	9	5
T14: 2.5 V Input Pulse Width, except PWRGOOD	2		BCLKs	8	Active and Inactive states
T14B: LINT[1:0] Input Pulse Width	6		BCLKs	8	7
T15: PWRGOOD Inactive Pulse Width	10		BCLKs	8 13	8

NOTES:
1. Not 100% tested. Specified by design characterization.
2. All AC timings for the CMOS signals are referenced to the BCLK rising edge at 0.7 V at the processor edge fingers. All CMOS signal timings (address bus, data bus, etc.) are referenced at 1.25 V at the processor edge fingers.
3. These signals may be driven asynchronously, but must be driven synchronously in FRC mode.
4. Valid delay timings for these signals are specified to 2.5 V +5%. See Table 3 for pull-up resistor values.
5. To ensure recognition on a specific clock, the setup and hold times with respect to BCLK must be met.
6. INTR and NMI are only valid during APIC disable mode. LINT[1:0]# are only valid during APIC enabled mode.
7. This specification only applies when the APIC is enabled and the LINT1 or LINT0 pin is configured as an edge triggered interrupt with fixed delivery, otherwise specification T14 applies.
8. When driven inactive or after Vcc_{CORE}, Vcc_{L2} and BCLK become stable.

3. FRCERR Pin Removed From Specification

The Pentium II processor will not use the FRCERR pin. All references to these pins will be removed from the specification. These references currently appear in the *Intel Architecture Software Developer's Manual, Volume 3: System Programming Guide*, Appendix B. These references also appear in the *Pentium® II Processor Developer's Manual*, Sections 3.2.7, 5.1.14, 7.5, 7.12 and A.1.23. Finally, these references appear in the *Pentium® II Processor at 233MHz, 266MHz, 300MHz, and 333MHz* datasheet, Sections 2.8, 2.13, and A.1.23.